La programmazione XML:

XSLT, SAX, DOM

Andrea Marchetti	IAT-CNR	Andrea.Marchetti@iat.cnr.it
Stefano Bovone	ELSAG-SPA	Stefano.Bovone@elsag.it
Massimo Martinelli	IEI-CNR	M.Martinelli@iei.pi.cnr.it

Programma

- Introduzione a XML
- Parser DOM e SAX
- XSLT

Metodi per elaborare documenti XML

Ogni passaggio sarà accompagnato da un esercizio

Introduzione a XML

Che cosa è XML

XML: acronimo di eXtensible Markup Language

- è un linguaggio estensibile realizzato per poter utilizzare in modo semplice i documenti strutturati
- studiato per il Web, possibilità di utilizzo in ambienti differenti.
- sviluppato dal W3C
- è un sottoinsieme di SGML

XML: gli obiettivi

Questi gli obiettivi progettuali di XML
secondo il W3C XML Working Group:

- XML deve essere utilizzato in modo semplice su Internet.
- XML deve supportare un gran numero di applicazioni.
- XML deve essere compatibile con SGML.
- Deve essere facile lo sviluppo di programmi che elaborino XML.
- Il numero di caratteristiche opzionali deve essere mantenuto al minimo possibile, idealmente a zero.
- I documenti XML dovrebbero essere leggibili da un uomo e ragionevolmente chiari.
- La progettazione XML dovrebbe essere rapida.
- La progettazione XML dovrebbe essere formale e concisa.
- I documenti XML devono essere facili da creare.
- Non è di nessuna importanza l'economia nel markup XML.

Markup
(marca, etichetta)

- *Markup*: tutto ciò che ha un significato speciale che deve essere ben caratterizzato

- Esempi di markup: *testo in corsivo*, <u>testo sottolineato</u>

- In XML tutto ciò che è compreso tra i caratteri "<" e ">" (*angled brackets, parentesi angolari*) è considerato *markup*, viene detto anche *tag* (etichetta), esempio:

  <nome>

- Anche HTML è un *markup language* inizialmente definito in SGML.

Limiti di HTML

Cosa è questo ?

```
<td> 12 </td>
```

- Il numero civico di una via ?
- Il numero di telefono per ottenere informazioni sugli abbonati ?
- Entrambe le cose ?
- Nessuna delle due ?

Un semplice markup con HTML

```
<p> <b> Sig. Mario Rossi </b>
<br>
Via Verdi, 12
<br>
56100, Pisa
```

Visualizzazione di markup HTML

Sig. Mario Rossi
Via Verdi, 12
56100, Pisa

Interpretazione di HTML

Il nostro algoritmo per trovare il numero civico:

Se un paragrafo contiene due tag

allora la prima parola dopo la prima virgola dopo il primo tag
 è il numero civico.

Un semplice markup XML

```
<business-card>
  <persona>
     <titolo> Sig. </titolo>
    <nome> Mario </nome>
    <cognome> Rossi </cognome>
  </persona>
  <indirizzo>
    <strada> Via Verdi </strada>
    <numero-civico> 12 </numero-civico>
    <cap> 56100 </cap>
    <città> Pisa  </città>
</business-card>
```

Visualizzazione di markup XML

Sig. Mario Rossi
Via Verdi, 12
56100, Pisa

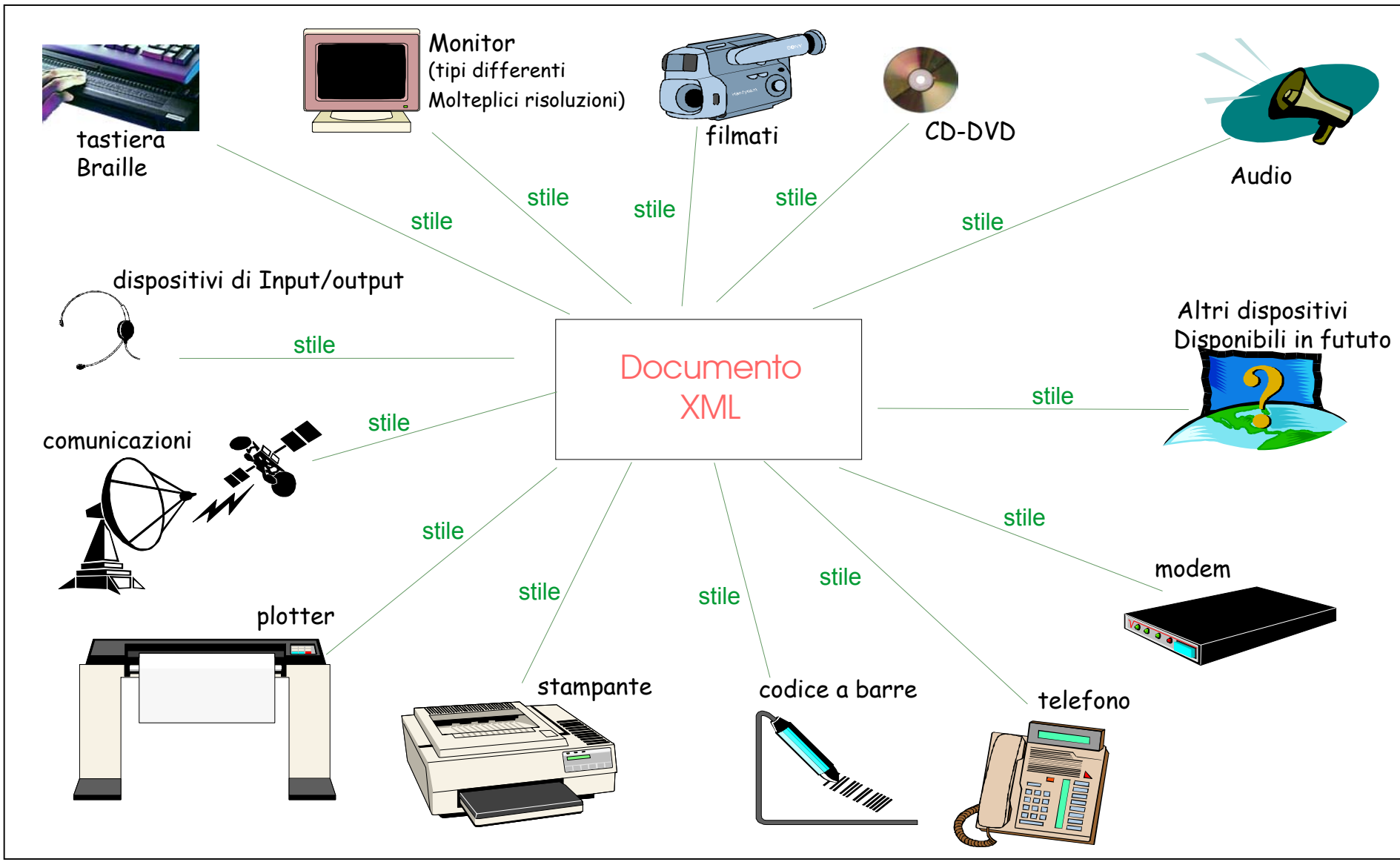
- XML può essere visualizzato nello stesso modo di HTML

Visualizzazione di markup XML

Il markup XML può essere visualizzato anche in questo modo:

Sig. Mario Rossi
Via Verdi, 12
56100, Pisa

Visualizzazione di markup XML



Interpretazione di XML

Un algoritmo migliore e più semplice per trovare il numero civico:

il numero civico è il contenuto del tag <numero-civico>

Contenuto contro rappresentazione

HTML ci dice come rappresentare un documento ipertestuale su Web (difficilmente su un altro media, ad esempio su carta)

XML ci dice cosa contiene un documento

Non è sufficiente migliorare HTML

Ricapitolando: limiti di HTML

- non ci dice nulla sul contenuto del documento
- non permette di estendere il linguaggio con tag personali
- limitato come prodotto di pubblicazione
- limitato come ipertesto
- limitato come elaborazione
- non supporta dati strutturati -> inefficiente per i motori di ricerca

Serve un linguaggio semplice, flessibile

HTML non verrà comunque sostituito, almeno nel più immediato futuro, perché offre il metodo più semplice per pubblicare informazioni sul Web

Le componenti di XML

Problema attuale: scambio di documenti
Formati proprietari difficilmente scambiabili

XML studiato per facilitare scambi di dati anche tra applicazioni di tipo diverso (es.: i database e i word processor).

Documento facilmente interpretabile
tre parti fondamentali da tenere distinte:

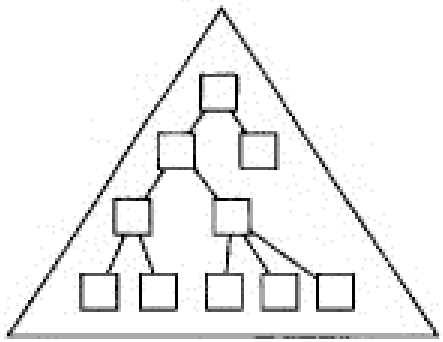
- il contenuto;
- le specifiche relative agli elementi, la struttura (DTD);
- le specifiche relative alla visualizzazione, lo stile (*Stylesheet*).

Le componenti di XML

Contenuto

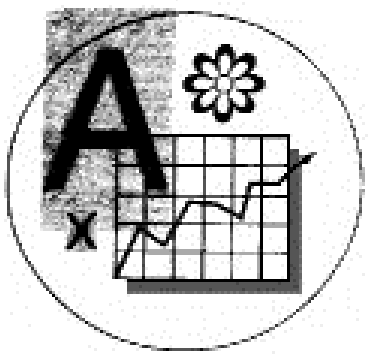
"La Discovery stava accelerando verso Giove lungo un'orbita complessa calcolata alcuni mesi prima dagli astronomi sulla Terra e controllata costantemente da Hal"
A.C. Clarke

Struttura



Specifiche di formattazione dell'output

Formattazione



Stile

Processo completo di codifica XML

Il documento

Uno degli obiettivi di progettazione di XML:
deve essere in un formato leggibile dall'uomo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<biblioteca>
  <libro codice="R414">
        <titolo>2001: Odissea nello spazio</titolo>
        <autore>
              <cognome>Clarke</cognome>
              <nome>Arthur Charles</nome>
        </autore>
        <editore>Rizzoli</editore>
        <parola_chiave>romanzo</parola_chiave>
        <parola_chiave>fantascienza</parola_chiave>
  </libro>
</biblioteca>
```

Il documento

- Tipicamente un documento inizia con una dichiarazione (anche se molti parser non la richiedono)

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

- L'attributo version è obbligatorio, se non è specificata la codifica (encoding) il valore di default è UTF-8

- Ciascun parser supporta un certo numero di codifiche

- XML è case sensitive

<nome> ≠ <Nome> ≠ <NOME>

Regole per i tag

Non permesso

```
<a>
    <b>
</a>
   </b>
```

Corretto

```
<a>
    <b>
    </b>
</a>
```

I tag devono essere nidificati

Quando l'elemento è senza contenuto
<tag attr="3"></tag>
<tag attr="3"/>
sono equivalenti

Nuova sintassi per i tag di chiusura

<tag attributo="valore">contenuto</tag>

Documento ben-formato, valido

Un documento XML si dice "ben formato" quando:
- contiene almeno un elemento;
- esiste un *tag* unico di apertura e di chiusura contenente l'intero documento;
- tutti i tag sono nidificati
- tutte le entità sono dichiarate.

Un documento si dice "valido" quando
- contiene una DTD e rispetta le regole definite in essa.

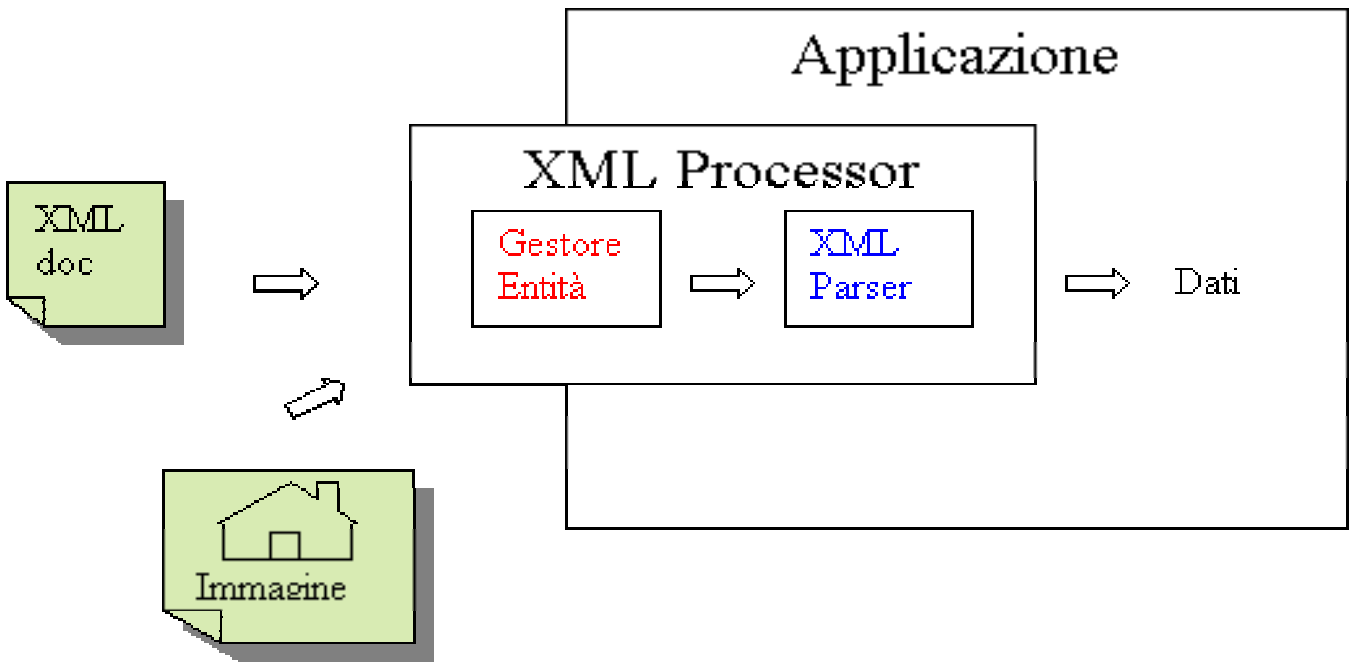
PARSER XML

Concetti di base

- Elaborare documenti/dati in formato XML
  - Generare documenti XML può essere molto facile
    ```
    system.out.println("<book><title>Moby Dick</title></book>");
    ```
  - Leggere e modificare non è altrettanto facile
- Necessità di librerie/pacchetti sw per la elaborazione XML
- XML Processor

XML Processor

- Architettura generale di un XML Processor



XML Processor

- XML Processor: sw che inserito in un'applicazione rende disponibile a tale applicazione il contenuto di un documento XML
- Compiti di un XML processor
  - gestione entità
  - validazione (parser)
- Importanza di una interfaccia standard tra applicazioni e XML processor
- Semplificare la sostituzione degli XML Processor

Creare un oggetto Parser

- La creazione di un oggetto Parser non è standard
- Ogni parser ha la sua implementazione
- Vediamo il codice dei parser più importanti
  - Apache (xerces)
  - Oracle
  - IBM (xml4j)
  - Sun
  - Microsoft (msxml)
  - ...

Xerces

- Fa parte di *Apache XML project*

```
import org.apache.xerces.parsers.* ...
try
{
    DOMParser myParser = new DOMParser();
    myParser.parse('document.xml');
}
```

Oracle

- Disponibile a oracle.com/xml [www.oracle.com/xml]

```
import oracle.xml.parser.v2.*
...
try {
    DOMParser myParser = new DOMParser();
      myParser.parse('document.xml');
}
```

XML4J-IBM

- Disponibile a www.alphaworks.ibm.com/tech/xml4j [www.alphaworks.ibm.com/tech/xml4j]

```
import com.ibm.xml.parsers.* ...
try
{
    DOMParser myParser = new DOMParser();
      myParser.parse('document.xml');
}
```

Prepariamo l'ambiente di lavoro

- Setting delle variabili di ambiente CLASSPATH e PATH
  - `set JDKDIR=c:\java\jdk1.1.8`
  - `set XML4JDIR=c:\java\xml4j_2_0_15`
  - `set CLASSPATH=%JDKDIR%\lib\classes.zip;%XML4JDIR&\xml4j.jar;.;`
  - `set PATH=%PATH%;.;%JDKDIR%\bin`
- Guarda nella directory *xmljava/ex1*
- Prova a compilare *cptest.java*
- *javac cptest.java*
- Se ottieni l'errore *Class not found* controlla se le variabili di ambiente sono impostate correttamente

Esercizio 1

- Creare l'albero DOM di un documento XML
- Guarda nella directory *xmljava/ex1*
- Troverai il file *createParser.java*
- Sostituisci *** con il tuo codice
- Il programma dovrebbe
  - creare un oggetto parser
  - fare il parsing del documento sonnet.xml
- Utilizziamo il parser Xerces
- La risposta è in *xmljava/ex1/soluzione*

Cosa abbiamo imparato

- Come creare un oggetto Parser di tipo DOM
- Come dire al parser di analizzare un documento XML e quindi creare l'albero DOM
- Ora dobbiamo vedere come
  - navigare
  - modificare
  - salvare
- l'albero DOM

DOM

Introduzione

- Indipendente dalla piattaforma e dal linguaggio
- La home page [http://www.w3.org/DOM] di DOM
- Disponibile per molti linguaggi Java, ECMA script (JavaScript, JScript), C++, Perl, Python, ...
- Implementato dalla maggior parte dei parser
- Sviluppato dal W3C in più passi o livelli

Attività al W3C

- Livello 1: contiene metodi per la navigazione e la manipolazione di documenti XML e HTML.
- Raccomandazione del 1 Ottobre 1998 [http://www.w3.org/TR/REC-DOM-Level-1]
- Livello 2: estende la elaborazione anche agli stylesheet e ai DTD. Candidato a Raccomandazione del 7 Marzo 2000 [http://www.w3.org/TR/DOM-Level-2]
- Livello 3: validazione tramite schema, specifiche per I/O. Requisiti del 20 Aprile 2000 [http://www.w3.org/TR/2000/WD-DOM-Requirements-20000412/#Level3]

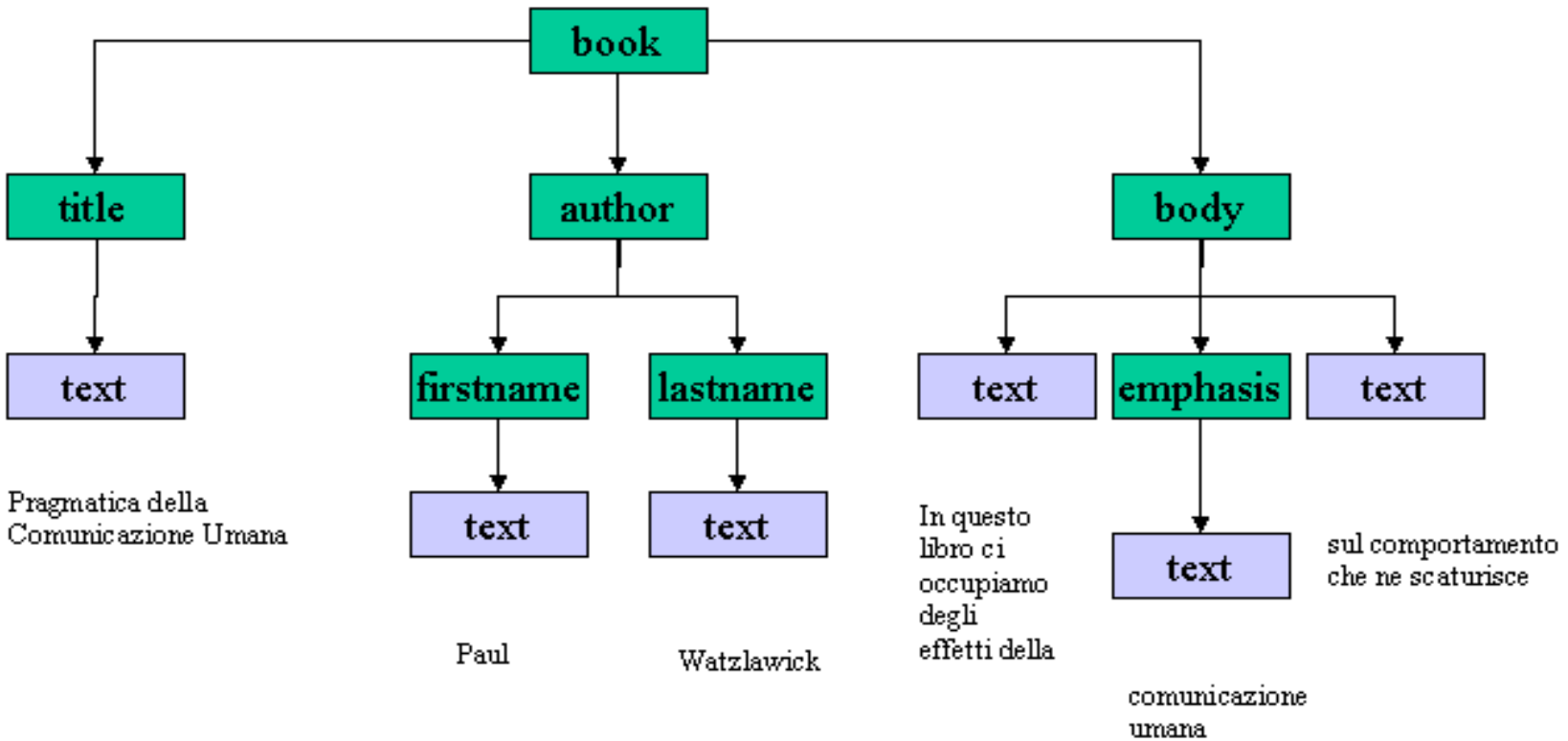
Perché si chiama DOM

- Definisce un modello ad *albero* del documento
- Consideriamo ad esempio il seguente documento XML
-

```
<?xml version='1.0'?>
<book><title>Pragmatica della Comunicazione Umana</title>
<author><firstname>Paul</firstname>
<lastname>Watzlawick</lastname></author>
<author><firstname>Janet Helmick</firstname>
<lastname>Beavin</lastname></author>
<author><firstname>Don D.</firstname>
<lastname>Jackson</lastname></author> <body>In questo
libro ci occupiamo degli effetti della
<emphasis>comunicazione umana</emphasis> sul
comportamento che ne scaturisce...</body> </book>
```

Modello ad Albero

- DOM rappresenta questo documento come il seguente albero



Gli oggetti DOM

```
<?xml version="1.0" ?>
<root>
        <hello>benvenuti al tutorial</hello>
</root>
```

Quanti figli ha l'elemento root?

Gli oggetti DOM

- DOM implementa l'albero tramite *oggetti*
- Modello ad Oggetti del Documento (D.O.M.)
- DOM assume di utilizzare un linguaggio O.O.
- Terminologia O.O.: interfacce, classi, oggetti
  - Il DOM definisce delle interfacce
  - I parser DOM implementano queste interfacce
  - Quando un parser analizza un documento XML crea degli oggetti
- Mischieremo l'uso dei termini, anche se presentando il DOM è corretto il termine interfaccia

La gerarchia DOM: I livello

- L'interfaccia *Node* è il cuore di tutto il DOM
- Quasi tutte le interfacce ereditano i metodi e le proprietà di *Node*
- La classe Node implementa metodi per
  - identificare il tipo di nodo (l'intero documento, un elemento, un attributo, ...)
  - navigare nell'albero DOM
  - modificare l'albero DOM

La gerarchia DOM: II livello

- *ELEMENT* (la maggioranza degli oggetti che utilizzerò)
- *ATTR* (l'attributo degli elementi, non fanno parte dei nodi dell'albero)
- *CHARACTERDATA* (il contenuto di un ELEMENT o di un ATTR)
- *DOCUMENT* (rappresenta l'intero albero XML)

Gerarchia DOM completa

- *Node*
  - *Attr*
  - CharacterData
    - Comment
    - *Text*
      - CDATASection
  - *Document*
  - *Element*
  - Notation
  - Entity
  - EntityReference
  - ProcessingInstruction
- *NodeList*
- NamedNodeMap

Document

- Rappresenta l'intero documento XML come un nodo
- Oggetto restituito dal parser dopo aver fatto la validazione Document doc = parser.getDocument();Rappresenta il punto di partenza di ogni applicazione DOM
- *Element root = Document.getDocumentElement();*//accesso al root Element
- *NodeList nl = Document.getElementsByTagName(String tagName);*//recupera tutti i nodi di tipo Element aventi un certo nome
- *Node newNode = Document.create...*//crea nuovi nodi di qualsiasi tipo (Element, Attr, Text, ...) conformi alla DTD associata al documento

Element

- Rappresenta un elemento XML come un nodo
- Eredita tutti i metodi della classe *Node*
- *String myTagName = Element.getTagName();*
  - restituisce il nome dell'elemento
- *NodeList nl = Element.getElementsByTagName();*
  - recupera tutti i nodi di tipo Element discendenti dall'elemento corrente, aventi un certo nome
- *String attrValue = Element.getAttribute(String attrName);*
  - recupera il valore di un attributo dell'elemento corrente
- *void Element.setAttribute(String attrName, String attrValue);*
  - inserisce il valore di un attributo
- Per operazioni più complesse sugli attributi conviene lavorare con le classi *Attr* e *NamedNodeMap*

Esercizio n.2

- Il programma deve
  - creare un oggetto parser
  - fare il parsing del documento book.xml
  - stampare il nome del tag dell'elemento root
  - stampare il valore dell'attributo 'id' dell'elemento root

Esercizio n.2

- Guarda nella directory *xmljava/ex2*
- Troverai il file *DomOne.java*
- Sostituisci *** con il tuo codice
- Utilizziamo sempre il parser Xerces
- La risposta è in *xmljava/ex2/soluzione*

Cosa abbiamo imparato

- Come accedere all'elemento radice di un albero DOM
- Come accedere ad un attributo di un elemento
- Una volta sulla radice possiamo navigare
- Occorrono i metodi dell'interfaccia *Node*

Metodi dell'interfaccia Node

- Identificare il tipo di nodo (l'intero documento, un elemento, un attributo, ...)
  - getNodeType() //Tipo di nodo come short
  - getNodeName() //Tipo di nodo come stringa
- I tipi di nodi sono
  - ELEMENT_NODE = 1;
  - ATTRIBUTE_NODE = 2;
  - TEXT_NODE = 3;
  - ...
  - DOCUMENT_NODE = 9;
  - ...

Metodi della classe Node

- la navigazione del modello della struttura DOM
  - hasChildNodes();
  - getChildNodes();
  - getFirstChild();
  - getLastChild();
  - getPreviousSibling();//Sibling=fratello/sorella
  - getNextSibling();
  - getParentNode()
- metodi per la sua modifica della struttura DOM
  - insertBefore(...);
  - replaceChild(...);
  - removeChild(...);
  - appendChild(...);
- Tali metodi vengono ereditati da tutte le altri classi che stanno sotto la gerarchia

NodeList

- Interfaccia che consente di gestire una lista di nodi
- I nodi sono acceduti tramite indice grazie ai metodi
  - *item()*
  - *getLength()*
- Esempio

```
NodeList nl= doc.getElementsByTagName('autore');
for (int i = 0 ; i < nl.getLength(); i++)
{
    Element el = (Element)(nl.item(i));
    // Elabora l'elemento el di tipo 'autore'
}
```

Navigazione

- In DOM per trovare gli elementi che ci interessano con i metodi forniti dalla specifica *Level 1* abbiamo 2 modi
  - *Navigazione a vista* si parte dalla radice con il metodo *getDocumentElement()* e muovendosi da nodo a nodo magari con procedure ricorsive
  - *Accesso diretto* si sfrutta il metodo getElementsByName(String tagName) presente nelle interfacce *Document* e *Element* che restituisce una *NodeList* di tipo element

Navigazione

- Nella navigazione a vista conviene:
  - sempre testare il tipo di nodo raggiunto
  - eseguire il relativo casting
  - applicare i metodi corrispondenti al tipo

```
if(nd.getNodeType()== ELEMENT_NODE)
   (Element)nd.getTagName()
```

- I due metodi possono essere usati insieme
- Se utilizziamo molto il DOM conviene crearsi una libreria di metodi per navigare

Alcuni consigli

- Quando si crea l'albero DOM di un documento XML gli errori classici sono
  - Il filename non è corretto
  - La DTD non può essere trovata
  - Il documento non è valido
  - Il documento non è well-formed
- Non dimenticare che il contenuto di un elemento è anch'esso un nodo (Text) dell'albero DOM

Esercizio n.3 Shakespeare Play

- Stampare un indice contenente i titoli degli atti e delle scene del documento Hamlet
- Guarda nella directory *xmljava/ex3*
- Troverai il file *Shakespeare.java*
- Sostituisci *** con il tuo codice
- La risposta è in *xmljava/ex3/soluzione*

Cosa abbiamo imparato

- I due metodi per navigare in DOM
- Come accedere al contenuto di un elemento
- Come scandire una NodeList

Aspetti da considerare

- La specifica DOM non tratta
  - Come accedere al documento generato dal Parser DOM *parser.getDocument()*
  - Come creare un documento DOM da zero
  - Come trasformare l'albero DOM in un documento XML
- Questi aspetti dipendono dalla particolare implementazione del Parser DOM
- Saranno oggetto delle specifiche a livello 3

Osservazioni

- Attenzione ai nodi testo creati da
  - spazi bianchi
  - line break
- Cose importanti non viste
  - Modifica della struttura di un albero DOM
  - Come convertire un albero DOM in documento XML
  - Come creare un albero DOM da zero

Conclusioni

- Una tecnologia presente da più di un anno
- Sono usciti vari prodotti che supportano DOM
  - Sito di Robin Cover presso OASIS [http://www.oasis-open.org/cober/dom.html]
  - XMLSoftware [http://www.xmlsoftware.com]
- DOM sarà incorporato negli editor XML per eseguire delle macro
- Si attendono a breve le estensioni DOM (level 2) per gestire gli stylesheet e i DTD
- Interazione con XQL, XPath, XPointer

SAX

Introduzione

- Sviluppato dai membri della lista XML-DEV@ic.ac.uk (non standard W3C)
- Adottato dalla maggioranza delle implementazioni dei parser/processor
- Disponibile in java, python, perl , C++

Storia

- Once Upon a Time (13 Dicembre 1997)
- Su iniziativa di Peter Murray-Rust
- Inizia una discussione con Peter Murray-Rust, Tim Bray, David Megginson
- La discussione si sposta su XML-DEV (David Megginson coordinatore)
- Sax 1.0 rilasciato 11 Maggio 1998
- Sax 2.0 rilasciato il 5 Maggio 2000?
- Sito ufficiale [http://www.megginson.com/SAX/index.html]

Modello di funzionamento: a eventi

- Il processor SAX legge il documento XML in *sequenza* come una stringa di caratteri
- Durante la lettura segnala certi *eventi*
- A questi eventi sono associate *azioni*
- Tu programmi queste azioni

Eventi SAX

- SAX definisce un certo numero di eventi associati alla lettura sequenziale dei componenti di un documento XML
  - Inizio e fine di un documento
  - Inizio e fine di un elemento
  - Contenuto di un elemento (Character Data)
  - ProcessingInstruction
  - Spazi bianchi ignorabili (linebreak, tab, space inseriti tra elementi che non hanno contenuto character data. Per questo motivo tali eventi sono segnalati solo se il documento contiene il riferimento al DTD)
  - Errori di sintassi XML di vario livello

Esempio

- Consideriamo il seguente esempio di documento XML **`<?xml version='1.0'><p>Hello, world!</p>`**
- Gli eventi generati da un processor SAX leggendo il documento sono
  - start document: *<?xml version='1.0'>*
  - start element: * *
  - characters: *Hello, world!*
  - end element: * *
  - end document

Associare un comportamento agli eventi

- Nelle applicazioni SAX tu decidi
  - *Quali* eventi gestire
  - *Come* gestire questi eventi
- SAX definisce quattro interfacce per gestire gli eventi, ognuna con un compito differente
  - DocumentHandler
  - ErrorHandler (gestisce gli errori contenuti nei documenti)
  - EntityResolver (gestisce i riferimenti ad entità esterne
  - DTDHandler (gestisce le definizioni di notazioni o entità contenute nel DTD)

Associare un comportamento agli eventi

- L'applicazione SAX deve informare il processore SAX su quali interfacce vuole gestire personalmente (normalmente le prime due) usando rispettivamente i seguenti metodi della classe SAXParser
  - SAXParser.setDocumentHandler(DocumentHandler handler)
  - SAXParser.setErrorHandler(ErrorHandler handler)
  - SAXParser.setEntityResolver(EntityResolver resolver)
  - SAXParser.setDTDHandler(DTDHandler handler)

Associare un comportamento agli eventi

- Un applicazione SAX non è obbligata a dichiarare tutti gli Handler
- Al limite può evitare di dichiarare dei suoi handler *accontentandosi* degli handler di *default* che il parser utilizza
- L'applicazione SAX per gestire le interfacce può
  - *implementare* completamente le singole *interfacce*
  - *estendere* la *classe* HandlerBase che le *implementa* tutte e quattro

Metodi interfaccia DocumentHandler

- **public void startDocument()**
- **public void endDocument()**
- **public void StartElement(String name, AttributeList attrs)**
- **public void EndElement(String name)**
- **public void characters(char ch[], int start, int length)**
- **public void ignorableWhitespace(char ch[], int start, int length)**
- **Controlla la documentazione per gli altri metodi**

Metodi interfaccia ErrorHandler

- **public void warning(SAXParseException exception)**
- **public void error(SAXParseException exception)**
- **public void fatalError(SAXParseException exception)**

Cosa conviene fare

- Se non si hanno esigenze particolare, la nostra applicazione SAX sarà una classe che :
  - estende la classe HandlerBase public class MyApplication extends HandlerBase
  - alcuni suoi metodi sostituiscono (override) i metodi del DocumentHandler
  - istanzia un oggetto parser

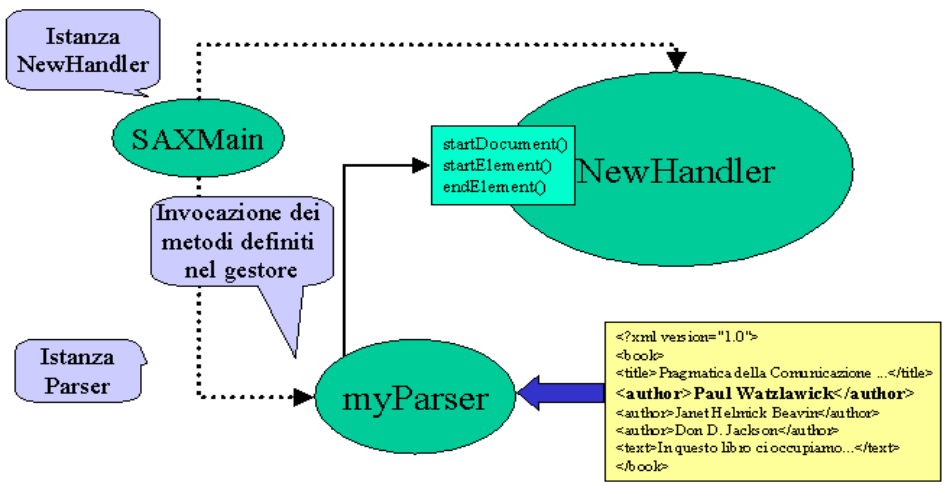
    ```
    SAXParser parser = new SAXParser();
    ```

  - dichiara al parser di essere il DocumentHandler

    ```
    parser.setDocumentHandler(this)
    ```

Architettura SAX

- Nel caso più generale l'implementazione del Handler viene incapsulato in una nuova classe



Esercizio 4

- Vogliamo imparare a ricevere e gestire gli eventi generati da un parser SAX
- Guarda nella directory *xmljava/ex4*
- Troverai il file *SaxOne.java*
- Sostituisci *** con il tuo codice
- Utilizziamo sempre il parser Xerces
- La risposta è in *xmljava/ex4/soluzione*

Esercizio 4

- Il codice completo dovrebbe analizzare un documento XML e stampare alcuni eventi SAX come essi occorrono
- Ricordarsi che i nomi delle classi Java sono case-sensitive
- Consideriamo il seguente documento xml

```
<?xml version='1.0'?>
<book> <title>Pragmatica della Comunicazione Umana</title> <author><firstname>Paul</firstname>
   <lastname>Watzlawick</lastname></author> <author><firstname>Janet Helmick</firstname>
   <lastname>Beavin</lastname></author> <author><firstname>Don D.</firstname>
   <lastname>Jackson</lastname></author>
<body>In questo libro ci occupiamo degli effetti della <emphasis>
comunicazione umana</emphasis> sul comportamento che ne scaturisce...</body>
</book>
```

- Vogliamo stampare i nomi degli autori
- Per eseguire l'applicazione dai il comando **java SaxOne book.xml**

Rivediamo il codice

- La classe *SaxOne* ha due metodi principali
  - *Elabora*
    - crea un parser SAX
    - stabilisce se stessa come *DocumentHandler*
    - esegue il parsing del documento XML
  - *main*
    - elabora la linea di comando
    - crea un oggetto *SaxOne*
    - passa il nome del documento XML all'oggetto *SaxOne*

Rivediamo il codice

- Gli altri metodi sono i metodi del DocumentHandler che vogliamo gestire personalmente
- import org.xml.sax.*;
  - pacchetto che contiene la SAX Api
- import org.apache.xerces.parsers.SAXParser;
  - Parser SAX di xerces

SAX Vs DOM

Primo esempio di applicazione

- Abbiamo la versione XML dell' *Iliade*
- Vogliamo stampare tutti i versi che citano *Agamennone*
- *Versione SAX*: implemento un metodo *startElement* in cui controllo quando siamo in un elemento *verso*, quindi implemento il metodo *character* per cercare la parola *Agamennone*
- *Versione DOM*: costruisco l'albero del documento quindi navigo attraverso i nodi alla ricerca della parola *Agamennone*

Primo esempio di applicazione

- In questo caso *SAX è più efficiente* in quanto dobbiamo accuparci di pochi elementi
- Usando DOM noi creiamo la struttura ad albero che utilizzeremo poco, inoltre visitiamo il documento due volte
  - una per creare l'albero
  - l'altra per cercare la parola

Sax è una buona scelta quando …

- Si deve attraversare il documento una sola volta
- Si stanno cercando pochi elementi
- La struttura del documento non è importante per l'applicazione
- Non si stanno cercando elementi dipendenti dal contesto
- La risorsa memoria è critica

Secondo esempio di applicazione

- Stiamo analizzando un documento contenente 10.000 indirizzi
- Vogliamo ordinarli per cognome
- *Versione SAX*: devo implementare i soliti eventi *startElement* e *character* in modo da memorizzare tutti gli indirizzi in una nostra struttura dati. Quindi eseguo l'ordinamento su questa struttura dati.
- *Versione DOM*: automaticamente costruisce la *struttura dati ad albero* del documento. Bisogna scrivere la routine di ordinamento sfruttando i metodi di accesso all'albero forniti dal DOM

Secondo esempio di applicazione

- In questo caso *DOM è più efficiente*
- Con SAX devo creare una struttura dati e dei metodi di accesso a questa struttura che con DOM sono gratuiti

DOM è una buona scelta quando …

- Si deve attraversare il documento più di una volta
- Si devono manipolare molte cose nel documento
- La struttura del documento è importante per l'applicazione
- Si stanno cercando elementi dipendenti dal contesto
- La risorsa memoria non è un problema

Riepilogo

- Due tipi di API XML:
  - Ad albero (DOM)
  - Ad eventi (SAX)
- API ad albero: Il parser memorizza tutto il documento xml in una struttura ad albero mettendo a disposizione dei metodi per navigarlo
- API ad eventi: Il parser mentre analizza il documento segnala degli eventi (inizio documento/elemento, fine documento/elemento ...).

Riepilogo

- Ad Albero
  - Molto potente grazie alla navigazione
  - Si può analizzare più di una volta
  - Grosso consumo di risorse (Memoria)
- Ad Eventi
  - Non richiede molte risorse di spazio e tempo
  - Accesso agli elementi a basso livello
  - Il documento si esamina tutto in una sola passata
- Un XML ad albero di solito per costruire la struttura interna utilizza un processor ad eventi
- Quando l'unico scopo dell'applicazione è quello di modificare la struttura di un documento XML, conviene considerare XSLT

Risparmiare tempo con gli alberi DOM

- Si può pensare che un *albero DOM* sia la versione compilata di un *documento XML*
- A volte conviene che due applicazioni si scambino direttamente l'albero DOM
- Un albero DOM occupa più spazio di un documento XML

Analisi di una stringa XML

- Se una applicazione genera una stringa XML e vuole fornirla ad un parser non ha bisogna di salvarla prima su di un file
- La maggioranza dei parser consentono di fare il parsing di un *InputSource*

Per convertire una stringa in un *InputSource*

```
StringReader sr = new StringReader ('<para>Hello,
  world!</para>');
InputSource xml = new InputSource(sr);
  parser.parse(InputSource);
```

XSLT

XML Transformations

- l' Extensible StylesheetLanguage per le trasformazioni (XSL-T) è una Raccomandazione del W3C [www.w3.org/TR/xslt] del 16 Novembre 1999
- XSLT consente di trasformare documenti XML in altri documenti XML (XHTML, WML)
- Le regole di trasformazione sono scritte in documenti XML detti *impropriamente* stylesheet
- XSLT può anche convertire un documento XML in un formato privo di markup (VRML)

Le basi di XSLT

- XSL-T usa i *templates* per specificare come gli elementi di un documento XML devono essere trasformati
- C'è un template per l'elemento *radice* del documento
- C'è un template di defualt per ogni elemnto che non è specificato
- Il processore XSL-T elabora il template dell'elemento *radice*
- Ogni elemento riferito dal template dell'elemento radice sarà elaborato a sua volta

Elementi di XSL-T

- **<xsl:template match="/">**
  - Template per l'elemento radice
- **<xsl:template match="x">**
  - Template per tutti gli elementi <x>
- **<xsl:template match="n/x">**
  - Template per tutti gli elementi <x> che stanno all'interno degli elementi <n>

Elementi di XSL-T

- **<xsl:apply-template select="x">**
  - Applica i templates a tutti gli elementi <x> contenuti nell'elemento corrente
- **<xsl:value-of select=".">**
  - Restituisce il valore dell'elemento corrente
- **<xsl:text>xyz</xsl:text>**
  - Restituisce in output il testo *xyz*

Elementi di XSL-T

- **<xsl:element name="a">**
  - Crea un elemento <*a*> e lo restituisce in uscita
- **<xsl:attribute name="href">**
  - Crea un attributo *href* su <xsl:element> corrente. Il contenuto di <xsl:attribute> diventa il valore dell'attributo creato

Elementi di XSL-T

- **<xsl:for-each select"x">**
  - Ciclo su tutti gli elementi <x> all'interno dell'elemento corrente
- **<xsl:if test="count(x)>1">**
  - Vero se c'è più di un elemento <x> all'interno dell'elemento corrente
- **<xsl:choose>, <xsl:when>, <xsl:otherwise>**
  - Sono gli equivalenti Java dei comandi *select*, *case* e *default*

Elementi di XSL-T

- **<xsl:variable>**
  - Consente di immagazzinare dati secondo delle regole di visibilità
- **<xsl:call-template name="x">**
  - Consente di invocare un template particolare
- **<xsl:with-param>**
  - Consente di passare dei parametri ad un template

Programmazione

– Da una applicazione si può richiamare un processore XSL per eseguire una trasformazione di un documento XML

```
XSLTProcessor processor =
  XSLTProcessorFactory.getProcessor();
processor.process(new XSLTInputSource(XMLFile),
        new XSLTInputSource(XSLFile),
        new XSLTResultTarget(NewXMLFile));
```

Bibliografia

- G. Ken Holman tutorial Practical Trasformations with XSL-T and XPath [http://www.CraneSoftwrights.com]
- Il capitolo 14 [http://metalab.unc.edu/xml/books/bible/updates/14.html] di XML Bible disponibile on-line

Dove reperire altre informazioni

Un elenco di indirizzi che può costituire un buon punto di partenza:

http://www.w3.org/XML (la home page di XML sul sito del W3C)

http://www.ucc.ie/xml (FAQ)
http://www.oasis-open.org/cover/sgml-xml.html (SGML/XML group)
http://www.microsoft.com/xml
http://www.mozilla.org
http://www.xml.org
http://www.xmlsoftware.com

Il Gruppo XML Italia

Il Gruppo XML Italia si è strutturato nell'ottobre del '98 come gruppo di lavoro aperto, basando le proprie procedure sul modello dei gruppi tecnici di Internet.

Scopo primario è quello di aggregare le competenze esistenti a livello nazionale per diffondere in modo organizzato la cultura XML

Le finalità principali:

- trasferire le conoscenze in ambito nazionale
- facilitare la discussione via e-mail e meeting periodici;
- essere un riferimento di informazioni su XML sia per gli sviluppatori sia per gli utenti;
- mostrare l'uso di queste nuove tecnologie tramite prototipi e semplici applicazioni.

Il Gruppo XML Italia

http://www.xml.it Il sito di XML Italia

Le mailing-list del gruppo XML-Italia:
org@xml.it riservata all'organizzazione
xml@xml.it aperta, dedicata alle attività tecnico-informative

consultabili anche su web:
http://listserv.xml.it/org.html
http://listserv.xml.it/xml.html