Tecnologie Cloud e Mobile

Lez. 03

XML

Giuseppe Psaila

Università di Bergamo

giuseppe.psaila@unibg.it

eXtensible Mark-up Language

- XML: eXtensible Mark-up Lamguage
  ovvero
  Linguaggio a Marcatori Estendibile
- XML è un linguaggio a marcatori generico:
- Non fornisce un insieme di marcatori predefinito (come HTML)
- Fornisce una struttura sintattica di base comune a tutti i linguaggi a marcatori

eXtensible Mark-up Language

- Fornisce gli strumenti linguistici per definire l’insieme di marcatori e le regole di correttezza sintattica
- In questo modo, si possono definire *classi di documenti XML*

Vantaggi

- XML è un formato aperto
- Non è legato a nessuna piattaforma hardware/software
- Adatto quindi per l’interscambio di documenti tra applicazioni diverse

Svantaggi

- Eccessiva “*verbosità*” del linguaggio
- Rispetto a un formato piatto e senza marcatori, un documento XML richiede molti più Byte
- La rappresentazione in memoria centrale richiede una struttura ad albero, la cui costruzione e navigazione non sono banali e richiedono maggiori risorse di calcolo

Esempio

```
<?xml version="1.0"?>
<!DOCTYPE PAPER SYSTEM "paper.dtd">

<PAPER name="sample doc">
  <TITLE> Using bibliography citations. </TITLE>
  <SECTION title="Introduction">
  This paper shows how to use a hypothetical
  <EMPH> XML language </EMPH> to describe structured
  documents which exploit the concept of citation
  </SECTION>
  <SECTION title="A Citation">
  Here we have an example of citation. We refer to
  Knuth's paper which introduced the concept of attribute
  grammar. This paper has the number <CITE label="Knuth68"/>
  in our bibliography.
  </SECTION>
```

Esempio

```
  <BIBLIOGRAPHY>
        <BIBITEM label="ASU85">
A.V. Aho, R. Sethi, J. D. Ullman, "Compilers: Principles,
Techniques, Tools", Addison-Wesley, 1985.
        </BIBITEM>
        <BIBITEM label="Knuth68">
D. E. Knuth, "Semantics of Context Free Languages",
Mathematical System Theory, Vol. 2, pp. 127-145, 1968.
        </BIBITEM>
  </BIBLIOGRAPHY>
</PAPER>
```

Elementi del Linguaggio

- Marcatori:
- Sono racchiusi tra “<” e “>”
- Delimitano gli elementi XML
- Marcatore di apertura:

  <Nome>

  <Nome *attributi*>

Elementi del Linguaggio

- Marcatore di chiusura:
  </Nome>
- Struttura a parentesi: ad ogni marcatore di apertura corrisponde un marcatore di chiusura con lo stesso nome.

<Nome>	<Nome *attributi*>
…	…
</Nome>	</Nome>

Elementi del Linguaggio

- Elemento (Element)
  Una coppia di marcatori di apertura e di chiusura descrive un *Elemento*
- Un elemento è un concetto semistrutturato, che ha un contenuto e delle proprietà (attributi)

Elementi del Linguaggio

- Esempio: SECTION di PAPER”

```
<SECTION title="A Citation">
 Here we have an example of citation. We
 refer to
 Knuth's paper which introduced the
 concept of attribute
 grammar. This paper has the number <CITE
 label="Knuth68"/>
 in our bibliography.
 </SECTION>
```

Elementi del Linguaggio

- Elemento Vuoto (Empty Element)
  Un elemento senza contenuto
- Può essere scritto nella forma compatta
  <Nome/>
  <Nome attributi/>
- Esempio:
  `<CITE label="Knuth68"/>`

Elementi del Linguaggio

- Attributo
  Una proprietà del concetto descritto dall'elemento
- Struttura:
  Nome = Valore
  dove valore è una stringa racchiusa tra virgolette ("...") o apici ('...')
- Nell'XML base non sono previsti tipi per gli attributi

Elementi del Linguaggio

- Esempio:

```
<SECTION title="A Citation">
<CITE label='Knuth68'/>
```

Elementi del Linguaggio

- Contenuto di un elemento
  Un misto di testo e occorrenze di elementi
  (contenuto semi-strutturato)
- Nell'XML base, non vi sono particolari vincoli sul contenuto dei singoli elementi.
  Questi vincoli possono essere specificati nel DTD (Document Type Definition).

Elementi del Linguaggio

- Marcatore di preambolo
  Tutti i documenti XML sono iniziati dal marcatore di apertura

```
<?xml version="1.0"?>
```

Elementi del Linguaggio

- È possibile specificare l'encoding del testo

```
<?xml version="1.0"
   encoding="ASCII"?>
<?xml version="1.0"
   encoding="UTF-8"?>
<?xml version="1.0"
   encoding="iso-8859-1"?>
```

Correttezza

- Il rispetto delle regole fino ad ora viste è un primo livello di correttezza dei documenti
- Un documento corretto da questo punto di vista viene detto

Ben Formato

DTD

- DTD: Document Type Definition
  Definisce la struttura dei documenti:
  - Elementi ammessi
  - Struttura del contenuto degli elementi
  - Attributi degli elementi

Meta-Tags

- `<!ELEMENT` *`Nome StrutturaCont.`*`>`
- *Nome*: il nome dell'elemento che si definisce
- *StrutturaCont.*: la struttura del contenuto dell'elemento

Meta-Tags

StrutturaCont.

- Un’espressione regolare che specifica la sequenza delle occorrenze degli elementi (nel contenuto)

Meta-Tags

StrutturaCont.

- Operatori:

(…)+	ripetizione non vuota
(…)*	ripetizione anche vuota
(…)?	opzionalità
E1,E2	sequenza di elementi
(E1 | E2 | …)	alternativa

Meta-Tags

StrutturaCont.

- Contenuto Testuale:
  (#PCDATA) solo testo
  (#PCDATA | E1 | E2 | …)*
  contenuto misto
- Contenuto vuoto:
  EMPTY

Meta-Tags

`<!ATTLIST` *`Elemento Nome Tipo Obblig.`*`>`

- *Elemento*
  per il quale si definiscono gli attributi
- *Nome*
  dell'attributo
- *Tipo*
  tipologia dell'attributo (non il tipo di dato)
- *Obblig.*
  opzione di obbligatorietà

Meta-Tags

Tipo	
CDATA	stringa generica
ID	identifica l’elemento
IDREF	riferimento all’ID di un altro elemento

Meta-Tags

Obblig.	
#REQUIRED	attr. obbligatorio
#IMPLIED	attr. facoltativo
valore_default	il valore da assumere quando non viene specificato

DTD per PAPER

```
<!ELEMENT BIBITEM (#PCDATA)>
<!ATTLIST BIBITEM label ID #IMPLIED>
<!ELEMENT BIBLIOGRAPHY (BIBITEM)+ >
<!ELEMENT CITE EMPTY >
<!ATTLIST CITE label IDREF #REQUIRED>
<!ELEMENT EMPH (#PCDATA) >
<!ELEMENT SECTION (#PCDATA | CITE | EMPH)*>
<!ATTLIST SECTION title CDATA #REQUIRED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT PAPER (TITLE,(SECTION)*,(BIBLIOGRAPHY)?)>
<!ATTLIST PAPER name ID #REQUIRED>
```

Elementi del Linguaggio

- Tipo del documento
  Nel documento XML si specifica il DTD,
  che definisce il tipo del documento
- Dopo il marcatore di preambolo

```
<!DOCTYPE PAPER SYSTEM "paper.dtd">
```

- Il file `"paper.dtd"` contiene il DTD

Correttezza

- Il rispetto delle regole definite nel DTD porta ad un più alto livello di correttezza dei documenti
- Un documento corretto da questo punto di vista viene detto

Valido

Parsing

Processore XML

- Secondo la specifica,
- Un «Processore XML» (o XML Processor)
- è un programma che ha il compito di processare un documento XML

Per Fare Cosa?

- Per utilizzare le informazioni riportate nel documento
- Al fine di elaborarle e svolgere delle attività specifiche

Parser?

- Un parser è uno strumento software che effettua l’analisi sintattica di testi
- Testi basati su un linguaggio artificiale
- Per esempio, per scrivere il compilatore di un linguaggio di programmazione occorre sviluppare il suo parser

XML Parser

- Anche XML è un linguaggio artificiale,
- Quindi, per poter processare un documento XML occorre un «XML Parser»
- Siccome il linguaggio ha una struttura sintattica comune, indipendente dalla particolare classe di documenti,
- esistono dei parser standard
- disponibili per i vari linguaggi di programmazione

Tipologie di Parsing

- SAX
- DOM

Parsing SAX

- Modalità detta a «Eventi»
- Quando viene attivato, al parser si deve fornire un oggetto «ContentHandler» (interfaccia)
- L’handler fornisce un metodo specifico per ogni elemento sintattico del linguaggio

Parsing SAX

- Per esempio, quando il parser incontra un marcatore di apertura, chiama un metodo specifico
- Quando trova un marcatore di chiusura, chiama un altro metodo

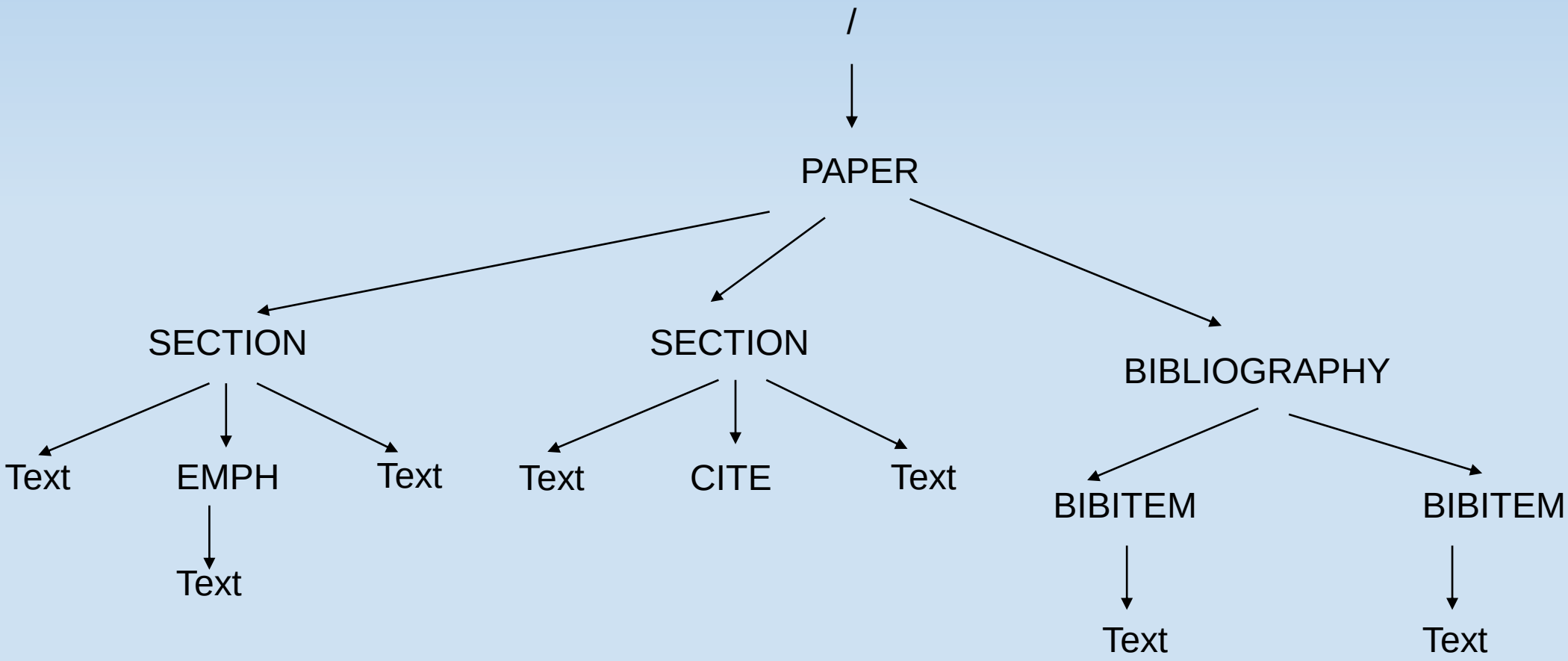
Esempio di Handler (Java)

```
public void startElement(
        String namespaceURI,
        String localName,
        String qName,
        Attributes atts) { … }
```

Parsing DOM

- Il parser costruisce una rappresentazione del documento in memoria centrale
- Esiste uno standard del W3C chiamato DOM
- Document Object Model

Struttura ad Albero



Nodi

I nodi dell’albero descrivono:

- Gli elementi
- I blocchi di testo
- Gli attributi degli elementi
  (nell’esempio, gli attributi non sono riportati)

Tipologie di Parser

Esistono tre tipologie di parser

- Non validante
  il parser verifica solo che il documento sia ben formato
- Validante
  Il parser verifica la correttezza del documento rispetto al DTD

Tipologie di Parser

- Validante rispetto a XML Schema
  È un’alternativa più moderna e flessibile al DTD
  Il parser valida il documento rispetto alla specifica XML Schema (ne parleremo più avanti)

XML e HTML

- A differenza di XML, HTML è un linguaggio a marcatori con un insieme ben definito di marcatori (elementi)
- Ma è pur sempre un linguaggio a marcatori, quindi, che relazione c'è?
- È possibile considerare HTML un caso particolare di XML?

Storia: SGML

- Il capostipite di XML e HTML è SGML
- SGML: Structured Generalized Mark-up Language
- Nato negli anni ‘80 nella comunità degli editori americani
- Per descrivere libri e articoli

Storia: SGML

- Era decisamente visionario
- Infatti prevedeva dei costrutti che non sono mai stati implementati da nessuno
- Perché erano troppo complicati (specie per le risorse computazionali dell'epoca)
- Tim Berners-Lee (l'inventore del World Wide Web) aveva lavorato al progetto SGML

Nascita di HTML

- Sapendo quali erano le potenzialità di unn linguaggio a marcatori
- Tim Berners-Lee decise di adottare questo approccio per definire HTML
- Partendo dalla sua esperienza con SGML
- Era il 1994 (nascita del World Wide Web)

Nascita di XML

- Nel 1996, dopo il successo di HTML
- Che tuttavia può solo descrivere le pagine web
- Nacque l’idea di un formato indipendente dal contesto applicativo
- Aperto e indipendente dalla piattaforma harware/software
- Nasceva XML

Nascita di XML

- XML doveva essere «il formato» dei dati sul web
- Deriva direttamente da SGML:
  - Ripulito dei costrutti impossibili
  - il parsing era diventato fattibile

Tuttavia

- Alcune scelte sintattiche di HTML non lo rendevano compatibile con XML
- Esempio:
-
 HTML
-
 XML

Adesso

- HTML 5 è compatibile con la sintassi di XML
- Anche se i browser sono tolleranti a scritture non propriamente «corrette»
- Il passo intermedio è stato XHTML, che poi è confluito in HTML 5

XML e JavaScript

JavaScript e XML

- Anche in JavaScript si possono elaborare documenti XML
- Si passa sempre attraverso la rappresentazione DOM
- Vediamo le caratteristiche del DOM attraverso l'implementazione di JavaScript

Gerarchia sui Nodi

- Lo standard W3C prevede una gerarchia sui nodi
- Questo approccio viene mantenuto anche in JavaScript
- Anche se, in realtà, il concetto di ereditarietà non esiste

Node

- Ogni pezzo (item) di un documento XML viene rappresentato da un oggetto Nodo (Node).
- Le relazioni di annidamento degli elementi nel documento XML,
  in DOM vengono rappresentate da una relazione padre-figlio tra nodi.
- Questo induce una struttura ad Albero

Node

- Tutti i nodi dell'albero sono definiti sulla classe Node
- In base al principio del Polimorfismo, Node viene specializzata in sottoclassi che corrispondono ai diversi tipi di item XML:
  - Element
  - Attribute
  - Text
  - …

Node

Campi:

- readonly NamedNodeMap **attributes**
- readonly NodeList **childNodes**
- readonly Node **firstChild**
- readonly Node **lastChild**

Node

Campi:

- readonly Node **nextSibling**
- readonly String **nodeName**
- readonly number **nodeType**
- String **nodeValue**

Node

Campi:

- readonly Document **ownerDocument**
- readonly Node **parentElement**
  [solo IE, incoerente]
- readonly Node **parentNode**
- readonly Node **previousSibling**

Node

Metodi:

- Node **appendChild**(Node newChild)
- Node **cloneNode**(boolean deep)
- Boolean **hasChildNodes**()
- Node **insertBefore**(Node newChild,
  Node refChild)

Node

Metodi:

- Node **removeChild**(Node oldChild)

- Node **removeNode**(
  boolean removeChildren)

- Node **replaceChild**(Node newChild,
  Node refChild)

Node e RecordType

- Il valore di questo campo indica il tipo di nodo.
- Ecco le costanti (di solito pre-definite):
  - **1 ELEMENT_NODE**
  - **2 ATTRIBUTE_NODE**
  - **3 TEXT_NODE**
  - **4 CDATA_SECTION_NODE**
  - **8 COMMENT_NODE**
  - **9 DOCUMENT_NODE**
  - **10 DOCUMENT_TYPE_NODE**
  - **11 DOCUMENT_FRAGMENT_NODE**

Element: sotto-classe di Node

- Estende Node
- Introducendo caratteristiche specifiche per descrivere gli Element

Element

Campi:

- readonly String **tagName**

  Per i nodi `Element`, coincide con `nodeName`

Element

Metodi:

- String **getAttribute**(String name)
- void **removeAttribute**(String name)
- void **setAttribute**(String name,
  String value)

Ritornano, cancellano e impostano un attributo (risp.)

Element

Metodi:

- NodeList **getElementsByTagName**(String name)

Fornisce la lista di nodi presenti nel sottoalbero con il tag name specificato.

CharacterData

- Il contenuto testuale dei documenti XML
- Questa classe è una sottoclasse di Node
- Qualsiasi pezzo di testo in mezzo a due marcatori viene rappresentato, anche gli a capo.

CharacterData

Campi:

- String **data**
- readonly number **length**

CharacterData

Metodi:

- void **appendData**(String arg)
- void **deleteData**(number offset, number count)
- void **insertData**(number offset, String arg)
- void **replaceData**(number offset, number count, String arg)
- String **substringData**(number offset, number count)

Text

- È una sotto-classe di CharacterData
- In effetti, i nodi testuali sono di tipo Text
- Aggiunge a CharacterData un metodo

  Text **splitText**(number offset)

  Il nodo è diviso, ma non si sa bene che cosa restituisca.

Document

- L’intero documento:
  Oggetto Document

- Contiene l’intero documento
- Consente di creare Elementi e Testi
- Consente di cercare gli Elementi
  - Per nome
  - Per identificatore

Document

Campi:

- readonly Element **documentElement**
  L'elemento radice del documento

Document

- Metodi:
- Element **createElement**(String tagName)
- Text **createTextNode**(String data)
- Element **getElementById**(String elementId)
- NodeList **getElementsByTagName**(
  String tagname)

NodeList

- Una classe aggiunta da DOM
- Che descrive liste di nodi
- Utile per rappresentare i nodi figli, oppure i nodi cercati con un certo nome

NodeList

- Campi:
- readonly number **length**
  elementi nella lista

- Metodi:
- Node **item**(number index)
  l'indice parte da 0

NamedNodeMap

- Descrive gli insiemi di attributi di un elemento
- Fornisce metodi per ottenere gli attributi in base a
  - Nome dell’attributo
  - Posizione

NamedNodeMap

Campi:

- readonly number **length**

NamedNodeMap

Metodi:

- Node **getNamedItem**(String name)
- Node **item**(number index)
  L'indice parte da 0
- Node **removeNamedItem**(String name)
- Node **setNamedItem**(Node arg)

NamedNodeMap

- Nota:
- I metodi restituiscono un Node
- Quindi, occorre poi gestore il contenuto dei nodi tramite gli appositi metodi

Creare Documenti XML

- Data la struttura DOM di un documento XML,
- per inviarlo, occorre **serializzarlo**,cioè generare il testo corrispondente.
- Gli interpreti JavaScript forniscono un oggetto *XMLSerializer*

Creare Documenti XML

Come usare XMLSerializer

- Dato un oggetto xml_doc (in DOM)

```
ser = new XMLSerializer();
doc = ser.serializeToString(xml_doc);
```

doc è la stringa con il testo XML generato

Creare Documenti XML

- Si può fare anche il processo di Parsing: da testo a DOM
- Si usa un oggetto DOMParser

```
var parser= new DOMParser();
var xmlobject  =
    parser.parseFromString(xmlstring, "text/xml");
```

Esempio di Scansione DOM

- Documento:

```
<?xml version="1.0"?>

<root>
  <name>A</name> <name>B</name>
</root>
```

Esempio di Scansione DOM

- Funzione JavaScript

```
function scandoc(doc)
{ var nl = doc.getElementsByTagName("name");
  var i;
  for(i=0; i < nl.length; i++)
```

Esempio di Scansione DOM

- Funzione JavaScript

```
for(i=0; i < nl.length; i++)
  {
      var n2 = nl.item(i).firstChild;
      myfunction( n2.data );
   }
}
```