Tehnici avansate de programare Curs 3

Cristian Frăsinaru

acf@infoiasi.ro

Facultatea de Informatică

Universitatea "Al. I. Cuza" laşi

Validarea documentelor XML

Cuprins

- Limbajul DTD
- Limbajul XML Schema
- Validarea documentelor XML în Java

Validarea documentelor XML

Ce este validarea?

Impunerea unor restricţii asupra unei baze de date care să asigure integritatea informaţiilor memorate.

Limbaje de validare pentru XML:

- DTD (Document Type Definition)
- XML Schema

Validarea nu este obligatorie.



Utilitatea validării

- Impunerea de restricţii asupra structurii unui document XML
- Modalitare de a defini conţinutul unui document XML
- Verificarea datelor de intrare ale aplicaţiilor



Clasificare documentelor XML

Dun punctul de vedere al validităţii, un document XML poate fi:

- bine-format respectă regulile generale de scriere în format XML dar nu are specificat un set de reguli de validare;
- valid respectă regulile generale şi formatul impus de un document de validare;
- invalid nu respectă regulile generale sau formatul impus de un document de validare.

Documentele valide sunt întotdeauna bine-formate, reciproca nefiind însă adevarată.

DTD

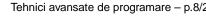
Ce este DTD?

Limbaj pentru validarea documentelor XML. Pune la dispoziţie modalităţi de a specifica care sunt tagurile permise şi cum pot fi acestea imbricate, care sunt atributele permise şi care sunt obligatorii, etc.

Regulile DTD pot fi declarate chiar în prologul documetului (inline) dar pot fi specificate şi extern.

Avantaj: orice document XML poate specifica intrinsec o descriere a formatului datelor pe care le conţine.

Dezavantaje: nu foloseşte sintaxa XML, are limitări majore.



Declarațiile regulilor

Declaraţia internă:

```
<!DOCTYPE radacina [
   Declaratii de reguli
]>
```

Declaraţia externă:

```
<!DOCTYPE radacina SYSTEM "fisier.dtd">
sau
<!DOCTYPE radacina SYSTEM "fisier.dtd" [
   Alte declaratii de reguli
]>
```

Sintaxa DTD

Din punctul de vedere al unui document de validare DTD un fişier XML poate fi format din următoarele:

- Elemente
- Atribute
- Entităţi
- Texte

Elemente

```
<!ELEMENT element categorie>
   sau
   <!ELEMENT element (continut)>
Categoriile unui element sunt definite prin:
```

- ANY
- EMPTY

```
<!ELEMENT b ANY>
<!ELEMENT i ANY>
<!ELEMENT br EMPTY>
```

Elemente cu conținut

```
<!ELEMENT element (#PCDATA)>
<!ELEMENT element (elemente-fiu)>
```

Caracter	Semnificaţie	Sintaxa
,	Enumerare	ELEMENT element (fiu1, fiu2,)
+	Unul sau mai mulţi	ELEMENT element (fiu+)
*	Zero sau mai mulţi	ELEMENT element (fiu*)
?	Opţional (zero sau unul)	ELEMENT element (fiu?)
	Sau	ELEMENT element (fiu fiu)

Exemplu

```
<!ELEMENT prezentare (slide+)>
<!ELEMENT slide (titlu, articol*)>
<!ELEMENT titlu (#PCDATA)>
<!ELEMENT articol (#PCDATA | articol)* >
```

Atribute

<!ATTLIST element numeAtribut tipAtribut valoareImplicita>

Tipul atributului	Ce reprezintă
CDATA	Tip caracter
(val1 val2)	O valoare din enumerarea specificată
ID	Un id unic
IDREF	ld-ul unui alt element
IDREFS	O listă separată de spații cu id-uri
NMTOKEN	Un nume XML valid
NMTOKENS	O listă de nume XML valide
ENTITY	O entitate
ENTITIES	O listă de entități
NOTATION	Numele unei notaţii
xml:	O valoare predefinită xml

Atribute (continuare)

Valoare	Ce reprezintă
valoare	Valoarea implicită
#REQUIRED	Atributul trebuie să fie obligatoriu prezent
#IMPLIED	Atributul poate să lipsească
#FIXED valoare	Valoarea atributului este fixă

Exemple

```
<!ELEMENT patrat EMPTY>
<!ATTLIST patrat latura CDATA "100">
<!ATTLIST persoana id CDATA #REQUIRED>
<!ATTLIST contact telefon CDATA #IMPLIED>
<contact telefon="123456"/> : declaratie valida
                            : declaratie valida
<contact/>
<!ATTLIST aplicatie limbaj CDATA #FIXED "Java">
<aplicatie limbaj="Java"/> : declaratie valida
<aplicatie limbaj="C++"/> : declaratie invalida
<!ATTLIST aplicatie limbaj CDATA (Java Python) "Java">
<aplicatie limbaj="Java"/> : declaratie valida
<aplicatie limbaj="Python"/> : declaratie valida
<aplicatie/>
                          : declaratie valida
<aplicatie limbaj="C++"/> : declaratie valida
```

Entități

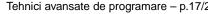
Entitățile permit definirea de constante de tip text. Enitățile interne vor fi definite astfel:

<!ENTITY entitate "valoare">

Referirea entitătilor se face prin expresii de tipul &entitate;

Exemplu:

```
<!ENTITY adr "http://java.sun.com">
<!ENTITY jdk  "Java 2 Standard Development Kit">
...
<text>
&jdk; poate fi descarcat de la adresa &adr;
Tot de la adresa &adr; poate fi consultata si documentatia.
</text>
```



Entități externe

<!ENTITY entitate SYSTEM "URI/URL">
Resursa externă poate referi orice URL valid:

Un fişier XML

```
<!ENTITY copyright SYSTEM "copyright.xml">
&copyright;
</text>
```

Un fişier DTD

```
<!ENTITY % entitati SYSTEM "fisier.dtd">
%entitati;
```

O componentă Web, cum ar fi un sevlet:

```
<!ENTITY currentDate SYSTEM
   "http://www.example.com/servlet/Today?fmt=dd-MMM-yyyy">
Data curenta este: &currentDate;
```

Exemplu de utilizare DTD

Un fisier XML valid

```
cprezentare titlu="Limbajul DTD">
  <slide tip="teorie">
    <titlu>Introducere</titlu>
    <articol>DTD = Document Type Definition</articol>
  </slide>
  <slide tip="teorie">
    <titlu>Sintaxa</titlu>
    <articol>
      <articol> Elemente </articol>
      <articol> Atribute </articol>
      <articol> Entitati </articol>
    </articol>
  </slide>
  <slide tip="exemplu">
    <titlu>Declaratii de elemente</titlu>
    <articol>
      <![CDATA[
        <!ELEMENT i ANY>
        <!ELEMENT br EMPTY>
      11>
    </articol>
  </slide>
</prezentare>
```

XML Schema

Ce este XML Schema?

XML Schema este alternativa bazată pe XML la DTD. Este referit şi sub denumirea de *XML Schema Definition* (XSD).

Avantajele acestuia sunt:

- Foloseşte sintaxa XML
- Este extensibil
- Oferă mai multe facilităţi
- Suportă tipuri de date
- Suportă spaţii de nume



Structura unui document XSD

```
<?xml version="1.0"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <!-- Declaratii de reguli -->
</xs:schema>
```

Referirea unei scheme dintr-un fişier XML se face astfel:

```
<?xml version="1.0"?>
<radacina xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="document.xsd">
    <!-- Continutul documentului -->
</radacina>
```

XML Schema este standard W3C:

http://www.w3.org/TR/xmlschema-0/

Tipuri de date XSD

Tipurile de date XSD se împart în următoarele categorii:

- Tipuri standard
- Tipuri simple
- Tipuri complexe

Cele mai comune tipuri standard de date sunt:

- xs:string
- xs:int
- xs:float
- xs:boolean
- xs:date



Tipuri simple

Un *element simplu* este un element ce conţine doar text - nu poate conţine alte elemente sau atribute.

Definirea unui element simplu se face astfel:

Exemple de tipuri simple

```
<nume>Popescu</nume>
<salariu>500</salariu>
<datan>1970-12-31</datan>
<conducere>true</conducere>

<xs:element name="nume" type="xs:string"/>
<xs:element name="salariu" type="xs:int" fixed="500"/>
<xs:element name="datan" type="xs:date"/>
<xs:element name="conducere" type="xs:boolean" default="false"/>
```

Atribute

Atributele trebuie definite ca tipuri simple. Doar tipurile complexe pot avea atribute.

Declararea unui atribut respectă următoarea sintaxă:

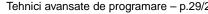
Exemple de atribute

Restricții

Restricțiile sunt folosite pentru a controla valorile pe care le poate primi un element sau atribut XML.

Faptul că un element sau atribut trebuie să respecte o anumită restricție va fi specificat astfel:

```
<xs:element name="numeElement" type="numeRestrictie">
<xs:attribute name="numeAtribut" type="numeRestrictie">
```



Tipuri de restricții

enumeration
fractionDigits
length
maxExclusive
maxInclusive
maxLength
minExclusive
minInclusive
minLength
pattern
totalDigits
whiteSpace

Exemple de restricții

```
<xs:restriction base="xs:string">
  <xs:enumeration value="Java"/>
  <xs:enumeration value="Python"/>
</xs:restriction>
<xs:restriction base="xs:float">
  <xs:minInclusive value="0"/>
  <xs:maxInclusive value="100"/>
</xs:restriction>
<xs:restriction base="xs:string">
  <xs:minLength value="5"/>
  <xs:maxLength value="8"/>
</xs:restriction>
<xs:pattern value="([a-z][A-Z])"/>
<xs:pattern value="([0-9])*"/>
<xs:pattern value="[a-zA-Z0-9]{8}"/>
<xs:pattern value="rosu|galben|albastru"/>
```

Tipuri complexe

Un *element complex* poate conţine alte elemente şi/sau atribute. Există patru tipuri de elemente complexe:

- elemente vide
- elemente ce conţin doar text
- elemente care conţin alte elemente
- elemente mixte, ce conţin atât text cât şi alte elemente

Fiecare dintre tipurile enumerate mai sus poate să conţină şi atribute.



Elemente vide

```
<xs:element name="numeElement">
  <xs:complexType>
    <xs:attribute name="numeAtribut" type="tipAtribut"/>
    . . .
    <!-- Nici o definitie de element -->
    </xs:complexContent>
  </xs:complexType>
</xs:element>
Exemplu:
<xs:element name="bursa">
  <xs:complexType>
    <xs:attribute name="tip" type="xs:string"/>
  </xs:complexType>
</xs:element>
<bursa tip="merit"/>
```

Elemente de tip text

```
<xs:element name="numeElement">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="tipDeBaza">
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>OR
sau
<xs:element name="numeElement">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="tipDeBaza">
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

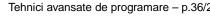
Exemplde de elemente de tip text



```
<xs:element name="nume"> <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="porecla" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
</xs:complexType> </xs:element>
<xs:element name="nota"> <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="xs:positiveInteger">
         <xs:minInclusive value="1" /> <xs:maxInclusive value="10" />
      </xs:restriction>
    </xs:simpleContent>
</xs:complexType> </xs:element>
Elementele de mai vor fi utilizate astfel:
  <nume porecla="iatagan">Turcu Gigel</nume>
  <nota>5</nota>
```

Elemente mixte

Definiția unui element mixt ce poate conține atât text cât și alte elemente setarea atributului mixed al tagului xsd:element la valoarea true:



Elemente formate din alte elemente

Un element poate fi declarat de tip complex astfel:

```
<xs:element name="numeElement" type="numeTipComplex">
```

Extinderea tipurilor

Putem defini tipuri complexe prin extinderea altor tipuri:

Indicatori

Indicatorii controlează modul de folosire a elementelor în cadrul documentului XML.

Tipurile de indicatori permişi sunt:

```
all
choice
```

sequence

minOccurs

max0ccurs

group

groupAttribute



Indicatori de ordine

Indicatori de apariții

Indicatori de grup

```
<xs:group name="numeComplet">
  <xs:sequence>
    <xs:element name="nume" type="xs:string"/>
    <xs:element name="prenume" type="xs:string"/>
  </xs:sequence>
</xs:group>
<xs:attributeGroup name="id info">
  <xs:attribute name="id" type="xs:int"/>
  <xs:attribute name="id sectie" type="xs:int"/>
</xs:qroup>
<xs:element name="persoana">
  <xs:complexType>
    <xs:all>
      <xs:group name="numeComplet">
      <xs:element name="salariu" type="xs:int"/>
    </xs:all>
    <xs:attributeGroup name="id info"/>
    <xs:attribute name="activ" type="xs:boolean"/>
  </xs:complexType>
</xs:element>
```

any, anyAttribute

Permit includerea de elemente/atribute care nu sunt specificate de schema de validare:

Referințe

Există posibilitatea de a defini independen un anumit element pentru ca apoi să-l refolosim în mai multe definiţii de tipuri complexe prin intermediul elementului ref.

Substituții

Substituţiile permit definirea de echivalenţe întele elementele unui text.

```
<xs:element name="persoana" type="xs:string"/>
<xs:element name="angajat" substitutionGroup="persoana"/>
<persoana> <nume>Ionescu</nume> </persoana>
<angajat> <nume>Popescu</nume> </angajat>
```

Prevenirea fenomenul de substituţie:

```
<xs:element name="persoana" type="xs:string" block="substitution"/>
```

Validarea documentelor XML folosind Java

Testarea validității

In ambele modele SAX şi DOM deosebim trei cazuri de testare a validității unui document XML:

- 1. Testarea doar a faptului că documentul este bine format.
- 2. Verificarea condițiilor impuse de reguli DTD.
- 3. Verificarea condiţiilor impuse prin scheme de tip XML Schema.



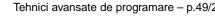
Validarea în modelul SAX

Documente bine-formate

Testarea doar a faptului că un document este bine-format se realizează prin inactivarea opțiunii de validare a parserului:

```
SAXParserFactory spf =
    SAXParserFactory.newInstance();
spf.setValidating(false);
```

Metoda parse va arunca excepţii de tip SAXException.



Tratarea excepţiilor SAXException

```
try {
 SAXParser saxParser = spf.newSAXParser();
 xmlReader = saxParser.getXMLReader();
 xmlReader.parse(new InputSource(filename));
} catch (SAXParseException e) { //Eroare de parsare
 System.err.println("Eroare de parsare"
   + ", linia " + e.getLineNumber()
   + ", fisierul " + e.getSystemId());
 System.err.println(e);
 System.exit(1);
} catch (SAXException e) { // Alte erori de tip SAXException
 System.err.println(e);
 System.exit(1);
System.err.println(e);
 System.exit(1);
```

Validare DTD

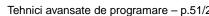
Trebuie să activăm opțiunea de validare:

```
SAXParserFactory spf =
    SAXParserFactory.newInstance();
spf.setValidating(true);
```

Nu vor mai fi generate excepţii!

Definirea unei clase de tip ErrorHandler ce va conţine metodele error, warning respectiv fatalError.

Erorile fatale vor fi generate la încălcarea regulilor de bază XML în timp ce erorile normale reprezintă nerespectarea regulilor de validare.



Definirea unui ErrorHandler

Definirea unei clase de tip ErrorHandler se va face astfel:

```
class MyErrorHandler implements ErrorHandler {
  public void error (SAXParseException e) {
    System.out.println("Eroare: " + e.getMessage());
  }
  public void warning (SAXParseException e) {
    System.out.println("Avertisment: " + e.getMessage());
  }
  public void fatalError (SAXParseException e) {
    System.out.println("Eroare fatala: " + e.getMessage());
    System.out.println("Nu mai putem continua...");
    System.exit(1);
  }
}
```

Asocierea dintre parser și handler

```
SAXParser saxParser = spf.newSAXParser();
xmlReader = saxParser.getXMLReader();

// Specficam objectul care se ocupa cu tratarea erorilor
xmlReader.setErrorHandler(new MyErrorHandler());

xmlReader.parse(new InputSource(filename));
```

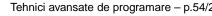
Interfața ErrorHandler este implementată și de clasa DefaultHandler.

Validarea XML Schema

Validarea folosind XML Schema necesită specificarea câtorva elemente adiţionale.

In primul rând trebuie să definim următoarele constante:

```
static final String JAXP_SCHEMA_LANGUAGE =
   "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
static final String W3C_XML_SCHEMA =
   "http://www.w3.org/2001/XMLSchema";
static final String JAXP_SCHEMA_SOURCE =
   "http://java.sun.com/xml/jaxp/properties/schemaSource";
```



Secvența generală de validare

```
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setNamespaceAware(true);
spf.setValidating(true);

SAXParser saxParser = spf.newSAXParser();
try {
    saxParser.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
} catch (SAXNotRecognizedException x) {
    // Va aparea exceptie daca parserul nu suporta JAXP 1.2
    ...
}
...
//Putem specifica un anumit fisier XML Schema
saxParser.setProperty(JAXP_SCHEMA_SOURCE, new File("fisier.xsd"));
```

Validarea în modelul DOM

Documente bine-formate

```
DocumentBuilderFactory dbf =
   DocumentBuilderFactory.newInstance();
dbf.setValidating(false);

DocumentBuilder db = dbf.newDocumentBuilder();
try {
   doc = db.parse(new File(filename));
} catch (SAXException e) {
   System.err.println(e);
   System.exit(1);
}
```

Validare DTD, XSD

```
DocumentBuilderFactory dbf =
  DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
dbf.setValidating(true);
try {
 dbf.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
} catch (IllegalArgumentException x) {
  // Va aparea exceptie daca parserul nu suporta JAXP 1.2
//Specificam fisierul XML schema
dbf.setAttribute(JAXP SCHEMA SOURCE, new File("fisier.xsd"));
DocumentBuilder db = dbf.newDocumentBuilder();
db.setErrorHandler(new MyErrorHandler());
```