# 35. MODELUL DE DATE XML

## 35.1 Documente XML

***XML*** (*eXtensible Markup Language*) este un *limbaj* generic bazat pe *marcatori* (taguri). Limbajul este considerat a fi *extensibil* deoarece permite utilizatorilor să îşi definească propriile elemente. XML a fost creat în principal pentru a facilita, prin intermediul Internetului, schimbul de date între diferite aplicaţii.

În continuare, este prezentat un document XML ce conţine o parte dintre angajaţii unei companii. Marcatorii poartă numele de ***elemente*** şi pot fi identificaţi cu uşurinţă deoarece sunt incluşi între semnele <>, ca de exemplu *Angajat*, *Nume*, *Prenume*. Marcatorii pot conţine şi o serie de opţiuni numite ***atribute***, cum ar fi *Departament*, *Cod*, *Varsta*. Alegerea modului în care sunt reprezentate datele în cadrul documentului (sub formă de elemente sau atribute) cade strict în sarcina dezvoltatorului.

```
<?xml version="1.0" standalone="yes"?>
<Angajati>
  <Angajat Departament="IT" Cod="100" Varsta="54">
    <Nume>Popa</Nume>
    <Prenume>Valeriu</Prenume>
  </Angajat>
  <Angajat Departament="DESFACERE" Cod="200" Varsta="20">
    <Nume>Ionescu</Nume>
    <Prenume>Ion</Prenume>
  </Angajat>
  <Angajat Departament="TRANSPORT" Cod="300" Varsta="65">
    <Nume>Popescu</Nume>
    <Prenume>Madalina</Prenume>
  </Angajat>
  <Angajat Departament="IT" Cod="400" Varsta="30">
    <Nume>Enescu</Nume>
    <Prenume>Adrian</Prenume>
  </Angajat>
  <Angajat Departament="CONTABILITATE" Cod="500" Varsta="40">
    <Nume>Vasile</Nume>
    <Prenume>Radu</Prenume>
  </Angajat>
</Angajati>
```

Corectitudinea unui document XML poate fi privită din două perspective diferite:

- din punct de vedere *sintactic* – fişierul trebuie să fie scris conform regulilor de sintaxă ale limbajului. De exemplu, fiecărui marcator de început trebuie sa-i corespundă un marcator de sfârşit;
- din punct de vedere *semantic* – conţinutul fişierului trebuie să fie valid, adică acesta trebuie să satisfacă anumite restricţii definite în schema documentului (DTD, XSD) sau chiar direct în fişier.

În cazul în care, pentru validare, se utilizează standardul DTD (Document Type Definition), acesta din urmă va trebui să definească structura documentului XML sub forma elementelor şi a atributelor considerate a fi valide. Astfel, grupuri de utilizatori pot agrea să folosească aceeaşi structură DTD pentru schimbul de date prin intermediul formatului

XML. De asemenea, pe baza DTD orice aplica□ie poate verifica validitatea datelor primite în format XML.

Exemplu de fişier DTD:

```
<!-- DOCUMENT TYPE DEFINITION -->

<!ELEMENT Angajati (Angajat+)>
<!ELEMENT Angajat  (Nume, Prenume)>
<!ATTLIST Angajat Cod CDATA #REQUIRED>
<!ATTLIST Angajat Departament CDATA #REQUIRED>
<!ATTLIST Angajat Varsta CDATA #IMPLIED>
<!ELEMENT Nume (#PCDATA)>
<!ELEMENT Prenume (#PCDATA)>
```

În documentul XML se va specifica numele fişierului DTD pe baza căruia se va realiza validarea:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Angajati SYSTEM "angajati.dtd">
<Angajati>
  <Angajat Departament="IT" Cod="100" Varsta="54">
    <Nume>Popa</Nume>
    <Prenume>Valeriu</Prenume>
  </Angajat>
  <Angajat Departament="DESFACERE" Cod="200" Varsta="20">
    <Nume>Ionescu</Nume>
    <Prenume>Ion</Prenume>
  </Angajat>
  <Angajat Departament="TRANSPORT" Cod="300" Varsta="65">
    <Nume>Popescu</Nume>
    <Prenume>Madalina</Prenume>
  </Angajat>
  <Angajat Departament="IT" Cod="400" Varsta="30">
    <Nume>Enescu</Nume>
    <Prenume>Adrian</Prenume>
  </Angajat>
  <Angajat Departament="CONTABILITATE" Cod="500" Varsta="40">
    <Nume>Vasile</Nume>
    <Prenume>Radu</Prenume>
  </Angajat>
</Angajati>
```

La fel de bine, descrierea DTD se poate include şi în cadrul documentului XML:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE ANGAJATI [
<!ELEMENT Angajati (Angajat+)>
<!ELEMENT Angajat  (Nume, Prenume)>
<!ATTLIST Angajat Cod CDATA #REQUIRED>
<!ATTLIST Angajat Departament CDATA #REQUIRED>
<!ATTLIST Angajat Varsta CDATA #IMPLIEDD>
<!ELEMENT Nume (#PCDATA)>
<!ELEMENT Prenume (#PCDATA)>
]>

<Angajati>
  <Angajat Departament="IT" Cod="100" Varsta="54">
    <Nume>Popa</Nume>
```

```
    <Prenume>Valeriu</Prenume>
  </Angajat>
  <Angajat Departament="DESFACERE" Cod="200" Varsta="20">
    <Nume>Ionescu</Nume>
    <Prenume>Ion</Prenume>
  </Angajat>
  <Angajat Departament="TRANSPORT" Cod="300" Varsta="65">
    <Nume>Popescu</Nume>
    <Prenume>Madalina</Prenume>
  </Angajat>
  <Angajat Departament="IT" Cod="400" Varsta="30">
    <Nume>Enescu</Nume>
    <Prenume>Adrian</Prenume>
  </Angajat>
  <Angajat Departament="CONTABILITATE" Cod="500" Varsta="40">
    <Nume>Vasile</Nume>
    <Prenume>Radu</Prenume>
  </Angajat>
</Angajati>
```

O altă modalitate de validare este reprezentată de folosirea unei scheme XSD (Xml Schema Definition) care va conţine descrierea structurii documentului XML.

Exemplu de fişier XSD:

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.tempuri.org/XMLSchema">

  <xs:element name="Angajati">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Angajat" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Angajat">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Nume" />
        <xs:element ref="Prenume" />
      </xs:sequence>
      <xs:attribute name="Varsta" type="xs:string"
use="optional" />
      <xs:attribute name="Departament" type="xs:string"
use="required" />
      <xs:attribute  name="Cod"  type="xs:string"  use="required"
/>
    </xs:complexType>
  </xs:element>


  <xs:element name="Nume">
    <xs:complexType mixed="true" />
  </xs:element>

  <xs:element name="Prenume">
    <xs:complexType mixed="true" />
  </xs:element>
```

```
</xs:schema>
```

Legătura dintre documentul XML si schema asociată se realizează astfel:

```
<?xml version="1.0" standalone="yes"?>
<Angajati xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="angajati.xsd">
  <Angajat Departament="IT" Cod="100" Varsta="54">
    <Nume>Popa</Nume>
    <Prenume>Valeriu</Prenume>
  </Angajat>
  <Angajat Departament="DESFACERE" Cod="200" Varsta="20">
    <Nume>Ionescu</Nume>
    <Prenume>Ion</Prenume>
  </Angajat>
  <Angajat Departament="TRANSPORT" Cod="300" Varsta="65">
    <Nume>Popescu</Nume>
    <Prenume>Madalina</Prenume>
  </Angajat>
  <Angajat Departament="IT" Cod="400" Varsta="30">
    <Nume>Enescu</Nume>
    <Prenume>Adrian</Prenume>
  </Angajat>
  <Angajat Departament="CONTABILITATE" Cod="500" Varsta="40">
    <Nume>Vasile</Nume>
    <Prenume>Radu</Prenume>
  </Angajat>
</Angajati>
```

De asemenea, schema XSD poate fi inclusă şi în cadrul documentului XML:

```
<?xml version="1.0" standalone="yes"?>
<Angajati>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Angajati">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Angajat" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Angajat">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Nume" />
        <xs:element ref="Prenume" />
      </xs:sequence>
      <xs:attribute name="Varsta" type="xs:string"
use="optional" />
      <xs:attribute name="Departament" type="xs:string"
use="required" />
      <xs:attribute  name="Cod"  type="xs:string"  use="required"
/>
    </xs:complexType>
  </xs:element>
```

```
  <xs:element name="Nume">
    <xs:complexType mixed="true" />
  </xs:element>

  <xs:element name="Prenume">
    <xs:complexType mixed="true" />
  </xs:element>

</xs:schema>

  <Angajat Departament="IT" Cod="100" Varsta="54">
    <Nume>Popa</Nume>
    <Prenume>Valeriu</Prenume>
  </Angajat>
  <Angajatt Departament="DESFACERE" Cod="200" Varsta="20">
    <Nume>Ionescu</Nume>
    <Prenume>Ion</Prenume>
  </Angajatt>
  <Angajat Departament="TRANSPORT" Cod="300" Varsta="65">
    <Nume>Popescu</Nume>
    <Prenume>Madalina</Prenume>
  </Angajat>
  <Angajat Departament="IT" Cod="400" Varsta="30">
    <Nume>Enescu</Nume>
    <Prenume>Adrian</Prenume>
  </Angajat>
  <Angajat Departament="CONTABILITATE" Cod="500" Varsta="40">
    <Nume>Vasile</Nume>
    <Prenume>Radu</Prenume>
  </Angajat>

</Angajati>
```

## 35.2 Modele XML

La prima vedere, modelul de date XML poate fi etichetat ca fiind unul foarte simplu. Cu toate acestea, el este în acelaşi timp considerat a fi şi foarte abstract deoarece oferă instrumentele necesare pentru construirea unor modele noi de o complexitate sporită.

De cele mai multe ori, un document XML poate fi văzut ca reprezentând liniarizarea unei structuri arborescente. Astfel, fişierul XML conţine nu numai înregistrările din arbore dar şi descrierea structurii acestuia, ca în figura 35.1.

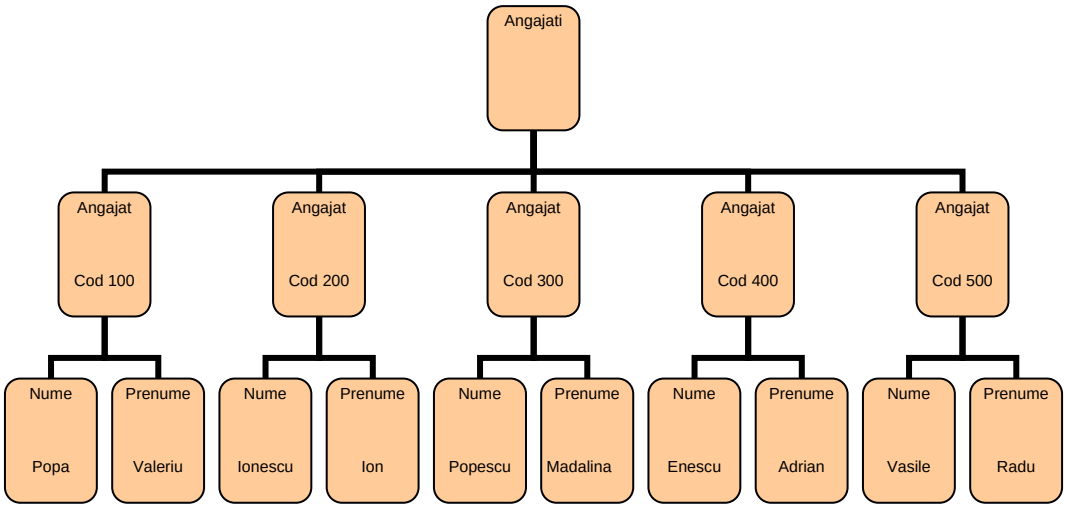


***Figura 35.1*** *Structura arborescentă corespunzătoare documentului XML*

Principala problemă cu care se confruntă aplicaţiile ce utilizează date de intrare structurate sub forma unor documente XML este aceea a parcurgerii corecte a structurii şi a extragerii informaţiilor necesare. Această operaţiune poartă numele de *parsare* şi poate fi realizată prin raportarea la două modele de bază:

- Document Object Model, DOM – documentul XML este itegral incărcat în memorie sub forma unui arbore ce poate fi parcurs şi examinat de către aplicaţie;
- Simple API for XML, SAX – este un model bazat pe evenimente (cum ar fi inceputul unui element sau atribut) ce sunt urmărite pe măsură ce documentul este parcurs. Nu necesită încărcarea întregului document în memorie iar datele sunt livrate pe măsură ce se înaintează în cadrul structurii XML.

Programul următor exemplifică încărcarea, parcurgerea şi afişarea sub formă arborescentă a unui document XML utilizând modelul DOM.

```
// XML DOM Object Model

#include "stdafx.h"
#include "filestream.h"

#define   RELEASE(p) {if (p) {(p)->Release(); p = NULL;}}


// verifica incarcarea documentului XML in memorie
HRESULT VerificaIncarcareDocumentXML(MSXML::IXMLDOMDocument* pDoc)
{
    MSXML::IXMLDOMParseError  *pXMLErr = NULL;
    LONG errCode = E_FAIL;

    pDoc->get_parseError(&pXMLErr);
    pXMLErr->get_errorCode(&errCode);
    if (errCode != 0)
    {
        fprintf(stderr, "documentul XML nu a putut fi incarcat\n");
    }
    else
    {
```

```
        fprintf(stderr, "documentul XML a fost incarcat cu succes\n");
    }

    RELEASE(pXMLErr);
    return errCode;
}


// parcurgerea in memorie a documentului XML sub forma arborescenta
HRESULT ParcurgeArboreXML(MSXML::IXMLDOMNode* node, int level)
{
    MSXML::IXMLDOMNode* pChild, *pNext;
    BSTR nodeName;
    MSXML::IXMLDOMNamedNodeMap* pattrs;

    node->get_nodeName(&nodeName);

    BSTR nodeText;
    node->get_text(&nodeText);

    for (int i = 0; i < level; i++)
    {
        printf(" ");
    }
    if (nodeName[0] != '#' || level == 0)
    {
        printf("%S",nodeName);
    }
    else
    {
        printf("%S", nodeText);
    }
    SysFreeString(nodeName);


    if (SUCCEEDED(node->get_attributes(&pattrs)) && pattrs != NULL)
    {
        pattrs->nextNode(&pChild);
        while (pChild)
        {
            BSTR name;
            pChild->get_nodeName(&name);
            printf(" %S='", name);
            ::SysFreeString(name);

            VARIANT value;
            pChild->get_nodeValue(&value);
            if (value.vt == VT_BSTR)
            {
                printf("%S", V_BSTR(&value));
            }
            printf("'");
            VariantClear(&value);
            pChild->Release();
            pattrs->nextNode(&pChild);
        }
        pattrs->Release();
    }
    printf("\n");

    node->get_firstChild(&pChild);
```

```
    while (pChild)
    {
        ParcurgeArboreXML(pChild, level+1);
        pChild->get_nextSibling(&pNext);
        pChild->Release();
        pChild = pNext;
    }

    return S_OK;
}


// incarca documentul in memorie sub forma arborescenta
HRESULT IncarcaDocumentXML(MSXML::IXMLDOMDocument *pDoc, BSTR pBURL)
{
    MSXML::IXMLDOMParseError  *pXMLError = NULL;
    VARIANT         vURL;
    VARIANT_BOOL    vb;
    HRESULT         hr;

    pDoc->put_async(VARIANT_FALSE);

    // Load xml document from the given URL or file path
    VariantInit(&vURL);
    vURL.vt = VT_BSTR;
    V_BSTR(&vURL) = pBURL;
    pDoc->load(vURL, &vb);

    hr = VerificaIncarcareDocumentXML(pDoc);

    RELEASE(pXMLError);

    return hr;
}


// conversie
BSTR ConversieAsciiCatreBSTR(const char* pszFName)
{
    WCHAR wszURL[MAX_PATH];
    ::MultiByteToWideChar(CP_ACP, 0, pszFName, -1, wszURL, MAX_PATH);
    return SysAllocString(wszURL);
}


// functia main
int _cdecl main(int argc, char *argv[])
{
    HRESULT hr = S_OK;
    MSXML::IXMLDOMDocument *pDoc = NULL;
    MSXML::IXMLDOMNode* pNode = NULL;
    BSTR pBURL = NULL;
    char* pszFileName = NULL;


    // initializare
    CoInitialize(NULL);

    // document XML gol
    CoCreateInstance(MSXML::CLSID_DOMDocument, NULL,
   CLSCTX_INPROC_SERVER,
```

```
                    MSXML::IID_IXMLDOMDocument, (void**)&pDoc);

    // argument - numele fisierului
    if (argc > 1)
    {
        pszFileName = argv[1];

        pBURL = ConversieAsciiCatreBSTR(pszFileName);

        IncarcaDocumentXML(pDoc, pBURL);

        hr = pDoc->QueryInterface(MSXML::IID_IXMLDOMNode,
(void**)&pNode);

        ParcurgeArboreXML(pNode,0);
    }

    RELEASE(pDoc);
    SysFreeString(pBURL);
    RELEASE(pNode);

    CoUninitialize();
    return 0;
}
```

Folosind ca parametru de intrare fişierul XML prezentat anterior, rezultatete obţinute în urma rulării programului sunt următoarele:

```
D:\>xmlDOM angajati.xml
documentul XML a fost incarcat cu succes
#document
 xml version='1.0' standalone='yes'
 Angajati
  Angajat Departament='IT' Cod='100' Varsta='54'
   Nume
    Popa
   Prenume
    Valeriu
  Angajat Departament='DESFACERE' Cod='200' Varsta='20'
   Nume
    Ionescu
   Prenume
    Ion
  Angajat Departament='TRANSPORT' Cod='300' Varsta='65'
   Nume
    Popescu
   Prenume
    Madalina
  Angajat Departament='IT' Cod='400' Varsta='30'
   Nume
    Georgescu
   Prenume
    Adrian
  Angajat Departament='CONTABILITATE' Cod='500' Varsta='40'
   Nume
    Vasile
   Prenume
    Radu
```

Următoarea funcţie este utilă în situaţia în care se doreşte salvarea sub forma unui fişier a documentului XML deja încărcat în memorie. Denumirea fişierului este transmisă sub formă de parametru.

```
// salveaza documentul XML aflat in memorie
HRESULT SalveazaDocumentXML(MSXML::IXMLDOMDocument *pDoc, BSTR
pBFName)
{
    HRESULT hr = S_OK;
    VARIANT vName;

    vName.vt = VT_BSTR;
    V_BSTR(&vName) = pBFName;
    hr = pDoc->save(vName);

    return hr;
}
```

Parcurgerea fişierului XML utilizând modelul SAX este exemplificată prin următorul program:

```
// SAX (Simple API for XML)

#include <xercesc/util/PlatformUtils.hpp>
#include <xercesc/util/TransService.hpp>
#include <xercesc/parsers/SAXParser.hpp>
#include "SAXPrint.hpp"
#include <xercesc/util/OutOfMemoryException.hpp>

static const char*              tipCodificare   = "LATIN1";
static XMLFormatter::UnRepFlags unRepFlags      =
XMLFormatter::UnRep_CharRef;
static char*  numeFisierXML   = 0;
static SAXParser::ValSchemes    schemaValidare  = SAXParser::Val_Auto;


int main(int argc, char* argv[])
{
    // initializare
    XMLPlatformUtils::Initialize();

    // preluare nume fisier
    // (parametru de intrare)
    numeFisierXML = argv[1];

    // obiectul SAX
    SAXParser* parser = new SAXParser;
    parser->setValidationScheme(schemaValidare);

    // parsare document XML (din fisier)
    int errorCode = 0;
    int errorCount = 0;
    try
    {
        SAXPrintHandlers handler(tipCodificare, unRepFlags);
        parser->setDocumentHandler(&handler);
        parser->setErrorHandler(&handler);
        parser->parse(numeFisierXML);
        errorCount = parser->getErrorCount();
    }
```

```
    catch (const OutOfMemoryException&)
    {
        XERCES_STD_QUALIFIER cerr << "Memorie insuficienta!!!" <<
XERCES_STD_QUALIFIER endl;
        errorCode = 1;
    }
    catch (const XMLException& toCatch)
    {
        XERCES_STD_QUALIFIER cerr << "\nEroare!\n  Descriere: "
             << StrX(toCatch.getMessage())
             << "\n" << XERCES_STD_QUALIFIER endl;
        errorCode = 1;
    }
    if(errorCode) {
        XMLPlatformUtils::Terminate();
        return errorCode;
    }

    // stergere obiect
    delete parser;

    XMLPlatformUtils::Terminate();

    return errorCount;
}
```

Rularea programului conduce la următoarele rezultate:

```
D:\>xmlSAX angajati.xml
<?xml version="1.0" encoding="LATIN1"?>
<Angajati>
  <Angajat Departament="IT" Cod="100" Varsta="54">
    <Nume>Popa</Nume>
    <Prenume>Valeriu</Prenume>
  </Angajat>
  <Angajat Departament="DESFACERE" Cod="200" Varsta="20">
    <Nume>Ionescu</Nume>
    <Prenume>Ion</Prenume>
  </Angajat>
  <Angajat Departament="TRANSPORT" Cod="300" Varsta="65">
    <Nume>Popescu</Nume>
    <Prenume>Madalina</Prenume>
  </Angajat>
  <Angajat Departament="IT" Cod="400" Varsta="30">
    <Nume>Georgescu</Nume>
    <Prenume>Adrian</Prenume>
  </Angajat>
  <Angajat Departament="CONTABILITATE" Cod="500" Varsta="40">
    <Nume>Vasile</Nume>
    <Prenume>Radu</Prenume>
  </Angajat>
</Angajati>
```

Indiferent de modelul ales (DOM sau SAX), efortul de programare rămâne considerabil deoarece dezvoltatorii trebuie să realizeze construcţii care să facă legătura între conţinutul documentului XML şi modul de reprezentare a acestuia la nivelul aplicaţiilor care îl folosesc.