Obsługa XMLa z poziomu Javy

Adam Kotwasiński

5 kwietnia 2009

Co możemy zrobić

- odczyt zawartości dokumentów XML
- modyfikacja i zapis dokumentów
- walidacja dokumentu
  - podczas parsowania
  - przed zapisaniem
  - względem DTD, XML Schema, innych standardów
- wsparcie dla innych standardów związanych z XML:
  - XSLT
  - XQuery, XPath
- XML w technologiach programistycznych
  - Web Services
  - AJAX

Abstrakcyjne modele dostępu

- korzystanie z gotowych parserów
  - brak konieczności ręcznej analizy warstwy leksykalnej
  - kontrola błędów składniowych
  - możliwość walidacji
- ustandaryzowany interfejs programistyczny
  - możliwość zmiany implementacji parsera
  - przenośność kodu
- różne modele dostępu
  - rozmiar
  - wymagane operacje
  - wymagana efektywność
  - dostępność schematu

XML i Java

- Ideologia
  - przenośność Javy
  - XML – międzyplatformowy nośnik danych
- Praktyka
  - Unicode i różne standardy kodowania
  - XML już w bibliotece standardowej – JAXP, JAXB

Klasyfikacja modelu

- Całość w pamięci
  - DOM – uniwersalny
  - JAXB – zależny od typu dokumentu
- Węzeł po węźle
  - SAX – model zdarzeniowy
  - StAX – przetwarzanie strumieniowe

Dokument w pamięci, interfejs uniwersalny

- Dokument reprezentowany przez drzewiastą strukturę danych
- Cechy charakterystyczne:
  - cały dokument w pamięci
  - jeden zestaw klas i metod
- Możliwe operacje
  - ładowanie dokumentu do pamięci
  - zapis dokumentu
  - chodzenie po drzewie
  - dowolna modyfikacja struktury
  - tworzenie nowych dokumentów

DOM Core

- Podstawowe metody dostępu do struktury dokumentu
  - umożliwia:
    - budowę dokumentów
    - nawigację po drzewie
    - dodawanie, modyfikację elementów i atrybutów
    - usuwanie elementów i ich zawartości
  - wady:
    - pamięć
    - niska efektywność
    - skomplikowany model dostępu do węzłów

Interfejs Node

- Dostęp do zawartości
  - getAttributes, getChildNodes, getFirstChild, getLastChild
  - getNextSibling, getPreviousSibling
  - getNodeName, getNodeValue, getNodeType
  - getOwnerDocument
  - getParentNode, hasChildNodes
- Manipulacja zawartością
  - appendChilde, insertBefore, removeChild, replaceChild
  - setNodeValue, setNodeName

DOM – style programowania

- jedynie interfejs Node:
  - nodeType
  - nodeName, nodeValue, childNodes
  - appendChild, removeChild
- interfejsy specyficzne dla rodzaju węzła
  - root – getDocumentElement
  - elementy – getElementsByTagName, getAttribute, setAttribute
  - atrybuty – getSpecified
  - wartości tekstowe – substringData, insertData

```
<?xml version="1.0"?>
<liczby>
<grupa wazne="tak">
<l>52</l><s>...</s>
</grupa>
<grupa wazne="nie">
<l>5</l><l>21</l>
</grupa>
<grupa wazne="tak">
<s>9</s><l>12</l>
</grupa>
</liczby>
```

```
int result = 0;
DocumentBuilderFactory factory =
  DocumentBuilderFactory.newInstance();
factory.setValidating(true);
DocumentBuilder builder =
  factory.newDocumentBuilder();
Document doc = builder.parse(args[0]);
Node cur = doc.getFirstChild();
while(cur.getNodeType() != Node.ELEMENT_NODE) {
  cur = cur.getNextSibling();
}
cur = cur.getFirstChild();
while(cur != null) {
  if(cur.getNodeType() == Node.ELEMENT_NODE) {
    String attVal = cur.getAttributes().
      getNamedItem("wazne").getNodeValue();
    if(attVal.equals("tak")) {
      result += processGroup(cur);
    }
  }
  cur = cur.getNextSibling();
}
```

```
private static int processGroup(Node group) {
int result = 0;
Node cur = group.getFirstChild();
while(cur != null) {
  if(cur.getNodeType() == Node.ELEMENT_NODE
  && cur.getNodeName().equals("l")) {
    StringBuffer buf = new StringBuffer();
    Node child = cur.getFirstChild();
    while(child != null) {
      if(child.getNodeType() == Node.TEXT_NODE)
        buf.append(child.getNodeValue());
      child = child.getNextSibling();
    }
    result += Integer.parseInt(buf.toString());;
    }
  cur = cur.getNextSibling();
  }
return result;
}
```

Wiązanie XML – idea

- Dokumenty XML a obiekty – klasa a schemat, obiekt a dokument
- Automatyczne generowanie klas ze schematów
- JAXB a DOM
  - zestaw klas i metod zależy od typu dokumentu
  - struktura mniej kosztowna pamięciowo
  - intuicyjny model dostępu do zawartości
  - modyfikacja struktury i wartości tylko w ramach tego samego typu dokumentu
- Implementacje – JAXB, Castor, Dynamic XML

Java API for XML Binding

- Standard opracowany przez Sun-a
- Zawarty w JSE 6
- Składniki:
  - definicja uniwersalnego fragmentu API
  - specyfikacja jak tłumaczyć
  - wsparcie dla XML Schema (obowiązkowe)

Używanie JAXB

1. Przygotowanie schematu dokumentów
2. Kompilacja narzędziem XJC
3. Napisanie aplikacji korzystając z:
   1. uniwersalnej części API JAXB
   2. klas wygenerowanych przez XJC

```
<xs:element name="liczby">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="grupa"
       minOccurs="0"
       maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name="l"
 type="xs:integer"/>
<xs:element name="s"
 type="xs:integer"/>
```

Wygenerowane klasy

```
Liczby
  List<Grupa> getGrupa()
```

Odpowiadające klasy

```
JAXBElement<BigInteger>
  QName getName()
  BigInteger getValue()
  void setValue(BigInteger)
```

```
<xs:element name="grupa">
<xs:complexType>
<xs:choice minOccurs="0"
maxOccurs="unbounded">
<xs:element ref="l"/>
<xs:element ref="s"/>
</xs:choice>
<xs:attribute name="wazne">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="tak"/>
<xs:enumeration value="nie"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element>
```

```
Grupa
  List<JAXBElement<BigInteger>> getLOrS()
  String getWazne()
  void setWazne(String)
```

```
int result = 0;
JAXBContext jc =
  JAXBContext.newInstance("jaxb_generated");
Unmarshaller u = jc.createUnmarshaller();
Liczby doc = (Liczby)u.unmarshal(
  new FileInputStream(args[0]));
List<Grupa> grupy = doc.getGrupa();
for(Grupa grupa : grupy) {
  if("tak".equals(grupa.getWazne())) {
    result += processGroup(grupa);
  }
}
```

```
private static int processGroup(Grupa aGrupa) {
int result = 0;
List<JAXBElement<BigInteger>> elems = aGrupa.getLOrS();
for(JAXBElement<BigInteger> elem : elems) {
  if("l".equals(elem.getName().getLocalPart())) {
    BigInteger val = elem.getValue();
    result += val.intValue();
  }
}
return result;
}
```

Model zdarzeniowy

- Kod wykonywany podczas czytania dokumentu
- Dokument XML jako ciąg zdarzeń (początek elementu, koniec dokumentu, itp.)
- Programista podaje kod, który wykona się w odpowiedzi na zdarzenia
- Treść dokumentu w parametrach
- Realizacje
  - obiekt obsługujący (handler) z odpowiednimi metodami
  - funkcje, wskaźniki do funkcji

Simple API for XML

- Wzorcowe interfejsy w Javie
- 1998 – SAX 1.0
- 2000 – SAX 2.0 – rozszerzenia:
  - przestrzenie nazw
  - cechy (features)
  - właściwości (properties) – dowolne obiekty

Implementacja

1. klasa implementująca interfejs ContentHandler
2. opcjonalnie ErrorHandler, DTDHandler, EntityResolver

Schemat aplikacji

1. pobranie XMLReader z fabryki
2. stworzenie obiektu obsługującego (handlera)
3. rejestracja obiektu obsługującego w parserze (XMLReader) metodami setContentHandler, setErrorHandler itp.
4. metoda parse
5. parser wykonuje nasz kod

```
private static class LiczbyHandler extends DefaultHandler {
  enum Stan {ZEWN, GRUPA, LICZBA};

  private int wynik = 0;
  private Stan stan = Stan.ZEWN;
  private StringBuffer buf;

  public int getResult() {
    return wynik;
  }

  public void startElement(String uri, String localName, String qName,
  Attributes attributes) throws SAXException {
    if("grupa".equals(qName)) {
      String attrVal = attributes.getValue("wazne");
      if("tak".equals(attrVal))
       stan = Stan.GRUPA;
    } else if("l".equals(qName)) {
      if(stan == Stan.GRUPA) {
        stan = Stan.LICZBA;
        buf = new StringBuffer();
} } }
```

```
public void characters(char[] ch, int start, int length)
throws SAXException {
  if(stan == Stan.LICZBA)
    buf.append(ch, start, length);
}

public void endElement(String uri, String localName, String qNam
throws SAXException {
  if("grupa".equals(qName)) {
    if(stan == Stan.GRUPA) {
      stan = Stan.ZEWN;
    }
  } else if("l".equals(qName)) {
    if(stan == Stan.LICZBA) {
      stan = Stan.GRUPA;
      wynik += Integer.parseInt(buf.toString());
} } } }
```

```
SAXParserFactory factory =
  SAXParserFactory.newInstance();
factory.setValidating(true);
SAXParser parser = factory.newSAXParser();
LiczbyHandler handler = new LiczbyHandler();
parser.parse(args[0], handler);
System.out.println("Result: "+handler.getResult());
```

Przestrzenie nazw

- bez obsługi
  - w fabryce ustawić setNamespaceAware(false)
  - budować drzewo parserem uzyskanym z takiej fabryki
  - używać metod w wersji bez końcówki NS
  - nie używać metod takich jak getNamespaceURI
- z obsługą
  - w fabryce ustawić setNamespaceAware(true)
  - budować drzewo parserem uzyskanym z takiej fabryki
  - używać metod w wersji z końcówką NS
  - identyfikować węzły na podstawie par: URI i lokalnej części nazwy

Filtry SAX

- implementacja interfejsu XMLFilter
  - zachowują się jak parser, ale źródłem danych jest inny XMLReader (parser lub filtr)
  - łańcuchy filtrów – metoda setParent
  - z punktu widzenia obiektu obsługującego filtr działa on jak parser (można wywołać jego metodę parse, on wywołuje metody obiektu obsługującego)
  - z punktu widzenia parsera zachowuje się on jak obiekt obsługujący — parser wywołuje metody filtra
- filtry pozwalają na:
  - filtrowanie zdarzeń
  - zmianę danych przed przesłaniem dalej
  - przetwarzanie dokumentu przez wiele modułów podczas jednego parsowania

```
XMLReader parser = XMLReaderFactory.getXMLReader();
ContentHandler handler = new MyHandler();
XMLFilter filtr = new MyFilter();
filtr.setParent(parser);
filtr.setContentHandler(handler);
filtr.parse();
```

```
public class LiczbyFiltr extends XMLFilterImpl {

private boolean czyPrzepuszczac = true;

public void characters(char[] aCh, int aStart, int aLength)
throws SAXException {
  if(czyPrzepuszczac)
    super.characters(aCh, aStart, aLength);
}

public void endElement(String aUri, String aLocalName,
String aName) throws SAXException {
  if(czyPrzepuszczac)
    super.endElement(aUri, aLocalName, aName);
  if("grupa".equals(aName))
    czyPrzepuszczac = true;
}

public void startElement(String aUri, String aLocalName,
String aName, Attributes atts) throws SAXException {
if("grupa".equals(aName) && "nie".equals(atts.getValue("wazne")))
  czyPrzepuszczac = false;
  if(czyPrzepuszczac)
    super.startElement(aUri, aLocalName, aName, atts);
} }
```

XSLT

- javax.xml.transform – przekształcenia dokumentów
- Transformer odpowiada definicji przekształcenia (np. arkusz XSLT)
- metoda transform
- wejście:
  - ciąg bajtów/znaków (np. plik, połączenie sieciowe)
  - strumień zdarzeń SAX
  - drzewo DOM
  - parser strumieniowy (od Java SE 6)
- transformacje identycznościowe zmieniające format a użycie filtrów

```
try {
  /* tworzymy transformer (z XSLT albo bez) */
  TransformerFactory trans_fact = TransformerFactory.newInstance();
  Transformer transformer;
  if(args.length >= 3)
    transformer = trans_fact.newTransformer(new StreamSource(args[2]));
  else
    transformer = trans_fact.newTransformer();

  /* zrodlo i cel przeksztalcenia */
  Source src = new StreamSource(args[0]);
  Result res = new StreamResult(args[1]);

  /* i przeksztalcamy */
  System.out.println("Poczatek przeksztalcania.");
  transformer.transform(src, res);
  System.out.println("Koniec przeksztalcania.");
} catch(Exception e) {
  System.out.println(e.getMessage());
  e.printStackTrace();
}
```

```
SAXParserFactory parser_fact =
  SAXParserFactory.newInstance();
XMLReader reader =
  parser_fact.newSAXParser().getXMLReader();

/* tworzymy transformer (bez XSLT) */
TransformerFactory trans_fact =
  TransformerFactory.newInstance();
Transformer transformer =
  trans_fact.newTransformer();

/* tworzymy filtr */
XMLFilter filtr = new NowyFiltr();
filtr.setParent(reader);

/* dokument, ktory bedziemy parsowac */
InputSource doc = new InputSource(args[0]);

/* teraz zrodlem zdarzen SAX dla Transformera jest filtr */
Source src = new SAXSource(filtr, doc);
Result res = new StreamResult(args[1]);

transformer.transform(src, res);
```

Walidacja

- javax.xml.validate – przekształcenia dokumentów
- może zmieniać strukturę dokumentu
- wynik tej samej postaci co wejście

```
SchemaFactory schemaFactory =
  SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA
Schema schemat =
  schemaFactory.newSchema(new StreamSource(args[1]));
Validator validator = schemat.newValidator();
validator.validate(new DOMSource(doc));

/* Zapisujemy zmieniony uzywajac Load and Save: */
DOMImplementationLS lsImpl =
  (DOMImplementationLS)domImpl.getFeature("LS", "3.0");
LSSerializer ser = lsImpl.createLSSerializer();
LSOutput out = lsImpl.createLSOutput();
out.setByteStream(new FileOutputStream(args[0]));

ser.write(doc, out);
```

XPath

- javax.xml.xpath
- dane – plik albo drzewo DOM
- wyrażenie w kontekście węzła – należy go podać jako węzeł DOM
- wynik tej samej postaci co wejście
- wynik – obiekt Javy typu najbardziej odpowiedniego dla wskazanego typu XPath (np. węzeł DOM)

```
XPath xpath =
  XPathFactory.newInstance().newXPath();
InputSource inputSource =
  new InputSource(args[0]);
NodeList nodes =
  (NodeList)xpath.evaluate(expr1,
     inputSource, XPathConstants.NODESET);
```

Model strumieniowy

- alternatywa dla modelu zdarzeniowego
- wyciąganie kolejnych zdarzeń z parsera – kontrola aplikacji, a nie parsera
- analogia do strumienia
- zachowane cechy SAX:
  - duża wydajność
  - dowolnie duże dokumenty
- standaryzacja
  - Common XmlPull API
  - Streaming API for XML

Model strumieniowy

- nowe korzyści:
  - przerwanie przetwarzania przed końcem pliku
  - szybsze filtrowanie
  - możliwość zmniejszenia liczby kopiowań napisów
- prosta obróbka wielu dokumentów jednocześnie
- mniej magiczny kod

StAX

- XMLStreamReader:
  - hasNext, next
  - getEventType, getName, getValue, getAttributeValue, itp.
- XMLEventReader
- XMLEvent
- XMLStreamWriter, XMLEventWriter
- XMLStreamFilter, XMLEventFilter

```
private static XMLStreamReader fReader;

public void run(String[] args) {
  int result = 0;
  XMLInputFactory factory = XMLInputFactory.newInstance();
  if(factory.isPropertySupported("javax.xml.stream.isValidating"))
    factory.setProperty("javax.xml.stream.isValidating", Boolean.FALSE);
  else
    System.out.println("walidacja nieobsługiwana");

  fReader = factory.createXMLStreamReader(new FileInputStream(args[0]));
  while(fReader.hasNext()) {
    int eventType = fReader.next();
    if(eventType == XMLStreamConstants.START_ELEMENT) {
      if(fReader.getLocalName().equals("grupa")) {
        String attrVal = fReader.getAttributeValue(null,"wazne");
        if("tak".equals(attrVal)) {
      result += this.processGroup();
  } } } }
  fReader.close();
  System.out.println("Result: "+result); }
```

```
private int processGroup() throws XMLStreamException {

int result = 0;

while(fReader.hasNext()) {
  int eventType = fReader.next();
  switch(eventType) {
    case XMLStreamConstants.START_ELEMENT :
    if("l".equals(fReader.getLocalName())) {
      String val = fReader.getElementText();
      result += Integer.parseInt(val);
    }
    break;
    case XMLStreamConstants.END_ELEMENT :
    if(fReader.getLocalName().equals("grupa")) {
      return result;
    }
    break;
  } }
  return result;
}
```

EMF

- Resource – plik, URI, itp.
- ResourceSet – przechowywanie związanych informacji
- niektóre konstrukcje XMLSchema nie są bezpośrednio mapowane do Ecore – metadane

EMF

- struktura:
  - root – nsURI, klasyfikatory
  - klasyfikatory – klasy i typy danych a complexType i simpleType
  - klasa ma nazwę i właściwości, obiekty klasy EObject
  - właściwości – inne obiekty (referencje), albo dane proste
  - refleksja – eGet(feature), eSet(feature)

```
<?xml version="1.0" encoding="UTF-8"?>
<tree:rootNode
 xmlns:tree="http://www.eclipse.org/emf/example/dom/Tree"
 label="root">
 <tree:childNode label="text">
 text
 </tree:childNode>
 <tree:childNode label="comment">
 <!--comment-->
 </tree:childNode>
 <tree:childNode label="cdata">
 <![CDATA[<cdata>]]>
 </tree:childNode>
</tree:rootNode>
```

EMF

1. utworzenie właściwości – korzeń
2. właściwość zawsze zdefiniowana wewnątrz klasy – dla roota specjalna klasa
3. tworzenie obiektu korzenia
4. wstawienie przestrzeni nazw
5. użycie fabryki do stworzenia obiektów typu złożonego (anyType to complexType)
6. dodanie atrybutu "label"
7. rozpoczęcie dodawania zawartości mieszanej

```
ResourceSet resourceSet = new ResourceSetImpl();
final ExtendedMetaData extendedMetaData =
  new BasicExtendedMetaData(
   resourceSet.getPackageRegistry());
resourceSet.getLoadOptions().put
  (XMLResource.OPTION_EXTENDED_META_DATA,
   extendedMetaData);

EStructuralFeature rootNodeFeature =
  extendedMetaData.demandFeature(NAMESPACE_URI,
  "rootNode", true);
EClass documentRootClass = rootNodeFeature.getEContainingClass();
EObject documentRoot = EcoreUtil.create(documentRootClass);

EMap xmlnsPrefixMap =
  (EMap)documentRoot.eGet
    (extendedMetaData.getXMLNSPrefixMapFeature(
       documentRootClass));
xmlnsPrefixMap.put(NAMESPACE_PREFIX, NAMESPACE_URI);

AnyType rootTreeNode = XMLTypeFactory.eINSTANCE.createAnyType();
documentRoot.eSet(rootNodeFeature, rootTreeNode);

EStructuralFeature labelAttribute =
  extendedMetaData.demandFeature(null, "label", false);
rootTreeNode.eSet(labelAttribute, "root");

FeatureMap rootMixed = rootTreeNode.getMixed();
```

EMF

1. utworzenie dziecka
2. dodanie atrybutu "label"
3. dodanie tekstu
4. pozostałe elementy analogicznie

```
EStructuralFeature childNodeFeature =
  extendedMetaData.demandFeature(NAMESPACE_URI,
                                 "childNode", true);
AnyType textChildTreeNode =
  XMLTypeFactory.eINSTANCE.createAnyType();
FeatureMapUtil.addText(rootMixed, "\n ");
rootMixed.add(childNodeFeature, textChildTreeNode);
textChildTreeNode.eSet(labelAttribute, "text");
FeatureMapUtil.addText(textChildTreeNode.getMixed(),
                       "text");
AnyType commentChildTreeNode =
  XMLTypeFactory.eINSTANCE.createAnyType();
FeatureMapUtil.addText(rootMixed, "\n ");
rootMixed.add(childNodeFeature, commentChildTreeNode);
commentChildTreeNode.eSet(labelAttribute, "comment");
FeatureMapUtil.addComment(
  commentChildTreeNode.getMixed(), "comment");
AnyType cdataChildTreeNode =
  XMLTypeFactory.eINSTANCE.createAnyType();
FeatureMapUtil.addText(rootMixed, "\n ");
rootMixed.add(childNodeFeature, cdataChildTreeNode);
cdataChildTreeNode.eSet(labelAttribute, "cdata");
FeatureMapUtil.addCDATA(
  cdataChildTreeNode.getMixed(), "<cdata>");
```

EMF

1. serializacja (GenericXMLResourceFactoryImpl od razu ustawia opcje dla dokumentu XML)
2. dodanie do zasobów
3. dodanie obiektu do zasobów
4. zapis
5. możliwa konwersja do DOM

```
resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put
  (Resource.Factory.Registry.DEFAULT_EXTENSION,
   new GenericXMLResourceFactoryImpl());
Resource resource =
  resourceSet.createResource(
    URI.createFileURI(
      DATA_FOLDER + "EMFDOMTreeNode.xml"));
resource.getContents().add(documentRoot);
resource.save(
  System.out, null);

Document document =
  ((XMLResource)resource).save(null, null, null);
TransformerFactory transformerFactory =
  TransformerFactory.newInstance();
Transformer transformer =
  transformerFactory.newTransformer();
transformer.transform(
  new DOMSource(document),
  new StreamResult(System.out));
System.out.println();
```

EMF

1. załadowanie zasobu
2. załadowanie korzenia – obiekt klasy AnyType
3. zapamiętanie możliwości dostępu do atrybutu "label" – możemy użyć tego samego API za każdym razem

```
ResourceSet resourceSet = new ResourceSetImpl();
final ExtendedMetaData extendedMetaData =
  new BasicExtendedMetaData(
    resourceSet.getPackageRegistry());
resourceSet.getLoadOptions().put
  (XMLResource.OPTION_EXTENDED_META_DATA,
   extendedMetaData);
resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put
  (Resource.Factory.Registry.DEFAULT_EXTENSION,
   new GenericXMLResourceFactoryImpl());

Resource resource =
  resourceSet.getResource(
    URI.createFileURI(
      DATA_FOLDER + "EMFDOMTreeNode.xml"), true);
EObject documentRoot =
  (EObject)resource.getContents().get(0);
AnyType rootTreeNode =
  (AnyType)documentRoot.eContents().get(0);

final EStructuralFeature labelAttribute =
  extendedMetaData.demandFeature(null, "label", false);
```

- rekurencyjne przejście
- wypisanie przestrzeni nazw, atrybutu ”label”, innych atrybutów

```
new Object()
{
 public void traverse(String indent, AnyType anyType)
 {
  System.out.println
    (indent + "{" + extendedMetaData.getNamespace(anyType.eContainmentFeature()) + "}" +
                 extendedMetaData.getName(anyType.eContainmentFeature()));
  System.out.println(indent + " label=" + anyType.eGet(labelAttribute));
  FeatureMap featureMap = anyType.getMixed();
  for (int i = 0, size = featureMap.size(); i < size; ++i)
  {
   EStructuralFeature feature = featureMap.getEStructuralFeature(i);
   if (FeatureMapUtil.isText(feature))
   {
    System.out.println(indent + " '" +
      featureMap.getValue(i).toString().replaceAll("\n", "\\\\n") "'");
   }
   else if (FeatureMapUtil.isComment(feature))
   {
    System.out.println(indent + " <!--" + featureMap.getValue(i) + "-->");
   }
   else if (FeatureMapUtil.isCDATA(feature))
   {
    System.out.println(indent + " <![CDATA[" + featureMap.getValue(i) + "]]>");
   }
   else if (feature instanceof EReference)
   {
    traverse(indent + " ", (AnyType)featureMap.getValue(i));
   }
  }
 }
}.traverse("", rootTreeNode);
```

- poprzedni przykład bardzo analogiczny do DOM
- metadane nie dostarczają bardzo użytecznych informacji
- xsd:anyType – możliwe jest utworzenie interfejsu AnyType przez Ecore-a, które daje nam dostęp do całej zawartości

```
<xsd:schema
 xmlns:tree="http://www.eclipse.org/emf/example/dom/Tree"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.eclipse.org/emf/example/dom/Tree">
 <xsd:element name="rootNode" type="xsd:anyType"/>
</xsd:schema>

<xsd:complexType name="anyType" mixed="true">
 <xsd:sequence>
  <xsd:any minOccurs="0" maxOccurs="unbounded"
   processContents="lax"/>
 </xsd:sequence>
 <xsd:anyAttribute processContents="lax"/>
</xsd:complexType>

public interface AnyType extends EObject
{
 FeatureMap getMixed();
 FeatureMap getAny();
 FeatureMap getAnyAttribute();
}
```

- wzbogacony schemat
- dodatkowy atrybut ecore:reference
- XML Schema to obiekt modelu i EMF dostarcza model XSD do reprezentacji tych schematów
- tworzenie takiego obiektu osiąga się analogicznie do AnyType

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
   xmlns:tree="http://www.eclipse.org/emf/example/dom/Tree"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.eclipse.org/emf/example/dom/Tree">
 <xsd:complexType mixed="true" name="TreeNode">
  <xsd:sequence>
   <xsd:element ecore:name="childNodes" form="qualified"
    maxOccurs="unbounded"
    name="childNode" type="tree:TreeNode"/>
  </xsd:sequence>
  <xsd:attribute name="label" type="xsd:ID"/>
  <xsd:attribute ecore:reference="tree:TreeNode" name="references">
   <xsd:simpleType>
    <xsd:list itemType="xsd:anyURI"/>
   </xsd:simpleType>
  </xsd:attribute>
 </xsd:complexType>
 <xsd:element name="rootNode" type="tree:TreeNode"/>
</xsd:schema>

XSDSchema xsdSchema = XSDFactory.eINSTANCE.createXSDSchema();
xsdSchema.setTargetNamespace(NAMESPACE_URI);
```

- analogiczne tworzenie instancji XSDEcoreBuilder
- każda przestrzeń nazw ma własny EPackage – kontener na wszystkie typy z tej przestrzeni nazw
- w przykładzie mamy tylko jedną przestrzeń nazw

```
XSDEcoreBuilder xsdEcoreBuilder =
new XSDEcoreBuilder(extendedMetaData);
URI schemaLocationURI =
  URI.createFileURI(new File(MODEL_FOLDER +
    "DOMEMFTreeNode.xsd").getAbsolutePath());
xsdEcoreBuilder.generate(schemaLocationURI);
EPackage ePackage =
  extendedMetaData.getPackage(NAMESPACE_URI);
```

- przejście po strukturze naszego schematu

```
System.out.println("package " + ePackage.getNsURI());
for (Iterator i =
ePackage.getEClassifiers().iterator(); i.hasNext(); )
{
 //
 EClassifier eClassifier = (EClassifier)i.next();
 if (eClassifier instanceof EClass)
 {
  //
  EClass eClass = (EClass)eClassifier;
  System.out.println(" class " + eClass.getName());
  for (Iterator j =
    eClass.getEStructuralFeatures().iterator(); j.hasNext(); )
  {
   //
   EStructuralFeature eStructuralFeature = (EStructuralFeature)j.next();
   if (eStructuralFeature instanceof EReference)
   {
    EReference eReference = (EReference)eStructuralFeature;
    System.out.println
     ("  reference " + eReference.getName() + ":"
       + eReference.getEReferenceType().getName());
   }
   else
   {
    EAttribute eAttribute = (EAttribute)eStructuralFeature;
    System.out.println
     ("  attribute " + eAttribute.getName() + ":"
       + eAttribute.getEAttributeType().getName());
   }
  }
 }
 else
 {
  EDataType eDataType = (EDataType)eClassifier;
  System.out.println(" data type " + eDataType.getName());
 }
}
```

- wynik

```
package http://www.eclipse.org/emf/example/dom/Tree
class DocumentRoot
 attribute mixed:EFeatureMapEntry
 reference xMLNSPrefixMap:EStringToStringMapEntry
 reference xSISchemaLocation:EStringToStringMapEntry
 reference rootNode:TreeNode
class TreeNode
 attribute mixed:EFeatureMapEntry
 reference childNodes:TreeNode
 attribute label:ID
 reference references:TreeNode
```

1. ładowanie obiektu
2. root traktowany jako TreeNode

```
Resource resource =
  resourceSet.getResource
    (URI.createFileURI(DATA_FOLDER +
    "EMFDOMTreeNode.xml"), true);

EObject documentRoot =
(EObject)resource.getContents().get(0);
final EObject rootTreeNode =
(EObject)documentRoot.eContents().get(0);

if (rootTreeNode.eClass() !=
extendedMetaData.getType(NAMESPACE_URI, "TreeNode"))
{
 throw new Exception("Bad meta data");
}
```

- zapamiętanie możliwości dostępu do właściwości

```
final EStructuralFeature mixedFeature =
  extendedMetaData.getMixedFeature(
    rootTreeNode.eClass());
final EStructuralFeature labelAttribute =
  extendedMetaData.getAttribute(
    rootTreeNode.eClass(), null, "label");
final EStructuralFeature referencesAttribute =
  extendedMetaData.getAttribute
    (rootTreeNode.eClass(), null, "references");
```

- użycie refleksji z EMF Object API

```
new Object()
{
 public void traverse(String indent, EObject eObject)
 {
  System.out.println
    (indent + "{" +
      extendedMetaData.getNamespace(eObject.eContainmentFeature())
      + "}" +
                  extendedMetaData.getName(eObject.eContainmentFeature()));
  System.out.println(indent + " label="
    + eObject.eGet(labelAttribute));

  ((List)eObject.eGet(referencesAttribute)).add(rootTreeNode);

  // Access the feature map reflectively.
  //
  FeatureMap featureMap = (FeatureMap)eObject.eGet(mixedFeature);
  for (int i = 0, size = featureMap.size(); i < size; ++i)
  {
   EStructuralFeature feature = featureMap.getEStructuralFeature(i);
   if (FeatureMapUtil.isText(feature))
   {
    System.out.println(indent + " '" +
      featureMap.getValue(i).toString().replaceAll("\n", "\\\\n") + "'");
   }
   else if (FeatureMapUtil.isComment(feature))
   {
    System.out.println(indent + " <!--" + featureMap.getValue(i) + "-->");
   }
   else if (FeatureMapUtil.isCDATA(feature))
   {
    System.out.println(indent +
      " <![CDATA[" + featureMap.getValue(i) + "]]>");
   }
   else if (feature instanceof EReference)
   {
    traverse(indent + " ", (EObject)featureMap.getValue(i));
   }
  }
 }
}.traverse("", rootTreeNode);
```

- bindowanie XML-i
- istnieją narzędzia do generowania API i implementacji
- DocumentRoot – metody do dostępu do obiektów globalnych
- użycie wygenerowanych rzeczy – rejestracja zasobów i pakietu
- refleksyjne API EMF-a dostarcza wielu użytecznych usług np. copier-a

```
public interface TreeNode extends EObject
{
 FeatureMap getMixed();
 EList getChildNodes();
 String getLabel();
 void setLabel(String value);
 EList getReferences();
}

public interface DocumentRoot extends EObject
{
 FeatureMap getMixed();
 EMap getXMLNSPrefixMap();
 EMap getXSISchemaLocation();
 TreeNode getRootNode();
 void setRootNode(TreeNode value);
}

resourceSet.getResourceFactoryRegistry()
.getExtensionToFactoryMap().put
 (Resource.Factory.Registry.DEFAULT_EXTENSION,
  new TreeResourceFactoryImpl());
resourceSet.getPackageRegistry().put
(TreePackage.eNS_URI, TreePackage.eINSTANCE);

Resource resource =
  resourceSet.getResource
    (URI.createFileURI(DATA_FOLDER +
      "EMFDOMTreeNodeWithReferences.xml"), true);

DocumentRoot documentRoot = (DocumentRoot)resource.getContents().get(0);
TreeNode rootTreeNode = documentRoot.getRootNode();

final EcoreUtil.Copier copier =
new EcoreUtil.Copier();
final DocumentRoot documentRootCopy =
  (DocumentRoot)copier.copy(documentRoot);
copier.copyReferences();
```

- przejście po drzewie
- każdy kopiowany węzeł ma referencje do oryginału

```
new Object()
{
 public void traverse(String indent, TreeNode treeNode)
 {
  System.out.println
    (indent + "{" +
      extendedMetaData.getNamespace(treeNode.eContainmentFeature()) +
      "}" +
                 extendedMetaData.getName(treeNode.eContainmentFeature()));
  System.out.println(indent + " label=" +
    treeNode.getLabel());
  for (Iterator i = treeNode.getReferences().iterator(); i.hasNext(); )
  {
   TreeNode referencedTreeNode = (TreeNode)i.next();
   System.out.println(indent + " reference=#"
     + referencedTreeNode.getLabel());
  }

  TreeNode treeNodeCopy = (TreeNode)copier.get(treeNode);
  treeNodeCopy.getReferences().add(treeNode);

  FeatureMap featureMap = treeNode.getMixed();
  for (int i = 0, size = featureMap.size(); i < size; ++i)
  {
   EStructuralFeature feature = featureMap.getEStructuralFeature(i);
   if (FeatureMapUtil.isText(feature))
   {
    System.out.println(indent + " '" +
      featureMap.getValue(i).toString().replaceAll
("\n", "\\\\n") + "'");
   }
   else if (FeatureMapUtil.isComment(feature))
   {
    System.out.println(indent +
      " <!--" + featureMap.getValue(i) + "-->");
   }
   else if (FeatureMapUtil.isCDATA(feature))
   {
    System.out.println(indent + " <![CDATA[" +
featureMap.getValue(i) + "]]>");
   }
   else if (feature instanceof EReference)
   {
    traverse(indent + " ", (TreeNode)featureMap.getValue(i));
   }
  }
 }
}.traverse("", rootTreeNode);
```

```
<?xml version="1.0" encoding="UTF-8"?>
<tree:rootNode
 xmlns:tree="http://www.eclipse.org/emf/example/dom/Tree"
 label="root"
 references="#root TreeNodeWithReferences.xml#root">
 <tree:childNode label="text"
  references="#root TreeNodeWithReferences.xml#text">
  text
 </tree:childNode>
 <tree:childNode label="comment"
  references="#root TreeNodeWithReferences.xml#comment">
<!--comment--></tree:childNode>
 <tree:childNode label="cdata"
  references="#root TreeNodeWithReferences.xml#cdata">
  <![CDATA[<cdata>]]>
 </tree:childNode>
</tree:rootNode>
```

- EMF – efektywny, wysokopoziomowy, abstrakcyjny model danych
- Ecore – prostszy niż Schema
- EObject – prostszy niż DOM
- inne:
  - cały zbiór danych jak DOM
  - wiązanie Schemy da coś w stylu bean-owatego API

Drzewo dokumentu

- wspólne cechy:
  - nieduże dokumenty
  - jednoczesny dostęp do wielu węzłów
  - tworzenie, edycja i zapis
- DOM
  - uniwersalność
  - obsługuje dokumenty bez schematów, lub ze schematami bardzo ogólnymi
- JAXB
  - ustalona struktura
  - komunikacja z aplikacją (np. wymiana danych)

Węzeł po węźle

- wspólne cechy:
  - duże dokumenty
  - proste, lokalne operacje
  - efektywność
- SAX
  - filtry
- StAX
  - przerwanie przetwarzania
  - prostszy (?) kod
  - równoległe przetwarzanie kilku plików

Bibliografia:

http://www.theserverside.com/tt/articles/article.tss?
track=NL-461&ad=647742HOUSE&l=BindingXMLJava&
asrc=EM_NLN_4036418&uid=2902821

http://www.mimuw.edu.pl/~czarnik/zajecia/xml08/index.html

Dziękuję za uwagę.