Document Object Model
w odniesieniu do XML

Marek Zawadka
deekay@gazeta.pl

Plan prezentacji

- Wstęp
- Krótko o XML
- Parsery – czym są, rodzaje, schemat użycia
- SAX
- DOM
  - Czym jest DOM?
  - Historia
  - Architektura
  - DOM Core
  - Porównanie DOM z SAX
- JDOM
- Testy wydajnościowe
- Podsumowanie
- Źródła

XML

- XML (*Extensible Markup Language*)
  - Podobny do HTMLa
  - Informuje o zawartości dokumentu a nie o tym jak go formatować

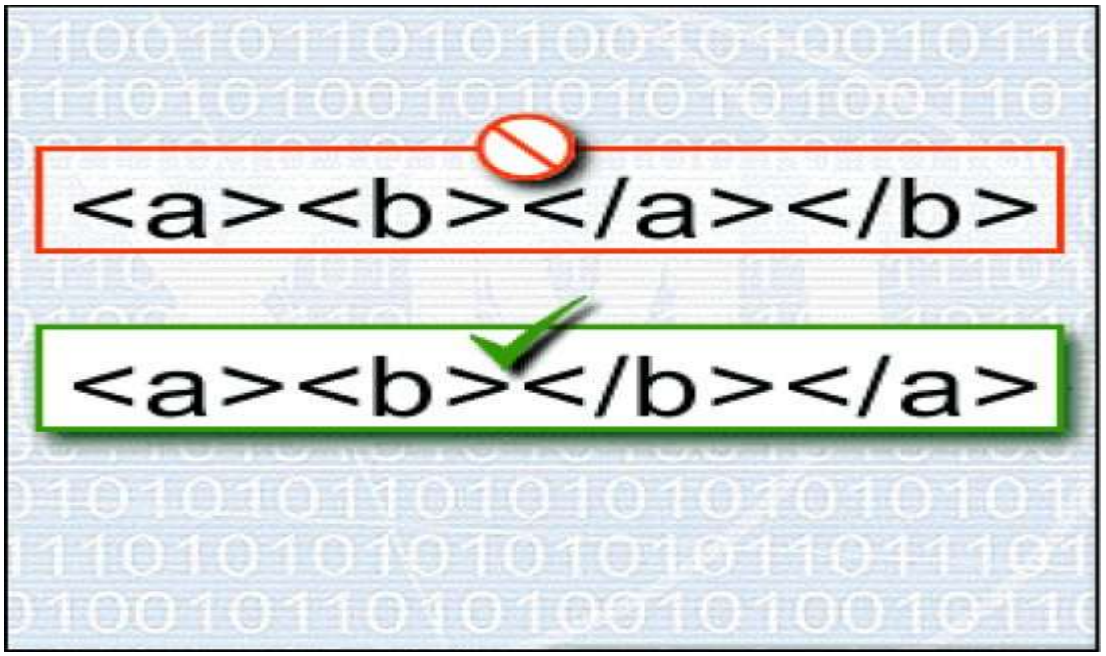
```
<person>
    <name first="Alan" last="Turing"/>
    <profession value="computer scientist"/>
    <profession value="mathematician"/>
    <profession value="cryptographer"/>
</person>
```

- DTD (*Document Type Definition*)
  - Pozwala samemu zdefiniować znaczniki dokumentu XML

```
<!DOCTYPE person [
  <!ELEMENT person(name+, profession*)>
  <!ELEMENT name EMPTY>
  <!ATTLIST name first CDATA #REQUIRED
                  last CDATA #REQUIRED>
  <!ELEMENT profession EMPTY>
  <!ATTLIST profession value CDATA #REQUIRED>
]>
```

Walidacja i "poprawne ukształtowanie"

- Dokumenty XML, które stosują się do pewnych zasad dotyczących tagów określamy jako `well-formed`



- Dokumenty XML, które używają DTD i stosują się do reguł tam zawartych określamy jako `valid`

Parser XML

- Coraz więcej aplikacji używa XML jako formatu kodowania danych
- Aby wydobyć z pliku potrzebne informacje (jeśli nie chcemy analizować pliku "ręcznie") potrzebujemy modułu, który będzie wykonywać dwie operacje
  - przeanalizuje tekst
  - przedstawi go w formie łatwej do przyswojenia przez aplikację
- Specyfikacja XML moduł taki nazywa *procesorem XML* (parserem)
- Można pokusić się o napisanie takiego parsera samemu jednakże:
  - nie jest to zadanie łatwe
  - istnieje sporo bardzo dobrych i darmowych programów i bibliotek do realizacji tej funkcji
- Schemat użycia parsera:

(1) Stwórz obiekt parsera

(2) Wskaż parserowi dokument XML

(3) Uzyskaj rezultaty

Podziały parserów

- Parsery walidujące oraz niewalidujące (*non-validating*)
- Parsery obsługujące DOM (Document Object Model)
- Parsery obsługujące SAX (Simple API for XML)
- Parsery napisane w poszczególnych językach (C++, Java, Perl itd.)

- Wszystkie parsery muszą sygnalizować błąd gdy wykryją, że dokument nie jest *well-formed*
- Parsery walidujące sprawdzają dokument podczas parsowania, zaś niewalidujące ignorują wszelkie błędy związane z walidacją
- Po co używać parserów niewalidujących? Odp: Szybkość działania
- Walidacja dokumentu jest kosztowna, jeśli jesteśmy przekonani o poprawności dokumentu dobrze jest jej unikać

Generowanie zdarzeń i SAX

- Parsery pracują na podobnej zasadzie – analizują dokument i po stwierdzeniu, że przeczytany ostatnio ciąg znaków układa się w pewną całość (np. znacznik początkowy) generują tzw. zdarzenia
- Zadaniem aplikacji jest dostarczenie funkcji, które będą wywoływane dla konkretnych zdarzeń
- W natłoku tego typu parserów ciężko było przenosić się na nowy-lepszy a jednocześnie niezgodny z poprzednikiem parser (zupełnie inne API)

- SAX jest próbą ujednolicenia interfejsu API dla tego typu parserów
- http://www.saxproject.org/
- Opracowany przez nieformalną grupę uczestników listy dyskusyjnej XML-DEV
- Dostępność: Java, C++, Perl, Python
- Aktualnie: SAX 2.0 (m.in. wsparcie dla przestrzeni nazw)

SAX

- Dokument jest przechodzony w całości od początku do końca
- Za każdym razem gdy parser “widzi” początek znacznika, jego koniec lub dane tekstowe mówi o tym programowi
- Brak dostępu do komentarzy dokumentu
- Dostęp jedynie do bieżącego fragmentu dokumentu (parser nie pamięta co było wcześniej ani nie wie co będzie potem)

org.xml.sax – pakiet ten zawiera kilkanaście interfejsów i klas, które można podzielić na kilka grup:

- interfejsy, które implementuje twórca parsera: `XMLReader, Attributes` i (opcjonalnie) `Locator`
- interfejsy implementowane przez autora aplikacji: `ContentHandler, DTDHandler, ErrorHandler`
- standardowe klasy SAX: `InputSource, SAXException, SAXNotRecognizedException, SAXNotSupportedException` i `SAXParseException`
- klasy pomocnicze dostępne w pakiecie org.xml.sax.helpers m.in. `DefaultHandler`

SAX

- XMLReader – najważniejszy interfejs z punktu widzenia SAX. Pozwala on zarejestrować obiekty obsługujące zdarzenia i rozpocząć przetwarzania dokumentu
- ContentHandler - jeśli aplikacja dostarczy implementacji metod tego interfejsu, wtedy będzie ona otrzymywać powiadomienia o wystąpieniu podstawowych zdarzeń
- DTDHandler – pozwala na dostęp do pewnych informacji w pliku DTD
- ErrorHandler - zawiera deklaracje metod wywoływanych w sytuacjach krytycznych
  - `public void error(SAXParseException e)`
  - `public void fatalError(SAXParseException e)`
  - `public void warning(SAXParseException e)`

- DefaultHandler = ContentHandler + DTDHandler + ErrorHandler

Interfejs ContentHandler

- W praktyce najczęściej wystarczy implementacja interfejsu *ContentHandler* i ewentualnie *ErrorHandler*

- startDocument()
- endDocument()
- startElement(String namespaceURI, String localName, String rawName, Attributes atts)

```
<a xmlns:h="http://www.my-namespace.pl">
  <h:b ala="ma kota">
</a>
```

- endElement(String namespaceURI, String localName, String rawName)
- characters(char[] ch, int start, int length)
- ignorableWhitespace(char[] ch, int start, int length)

SAX – przykład (1/3)

- XML

```
<?xml version="1.0"
    encoding="iso-8859-2"?>
    <autor>
    <imię>Paweł</imię>
    <nazwisko>Nieznany
    </nazwisko>
    <e-mail>noname@wp.pl
    </e-mail>
    </autor>
```

- Wyjście

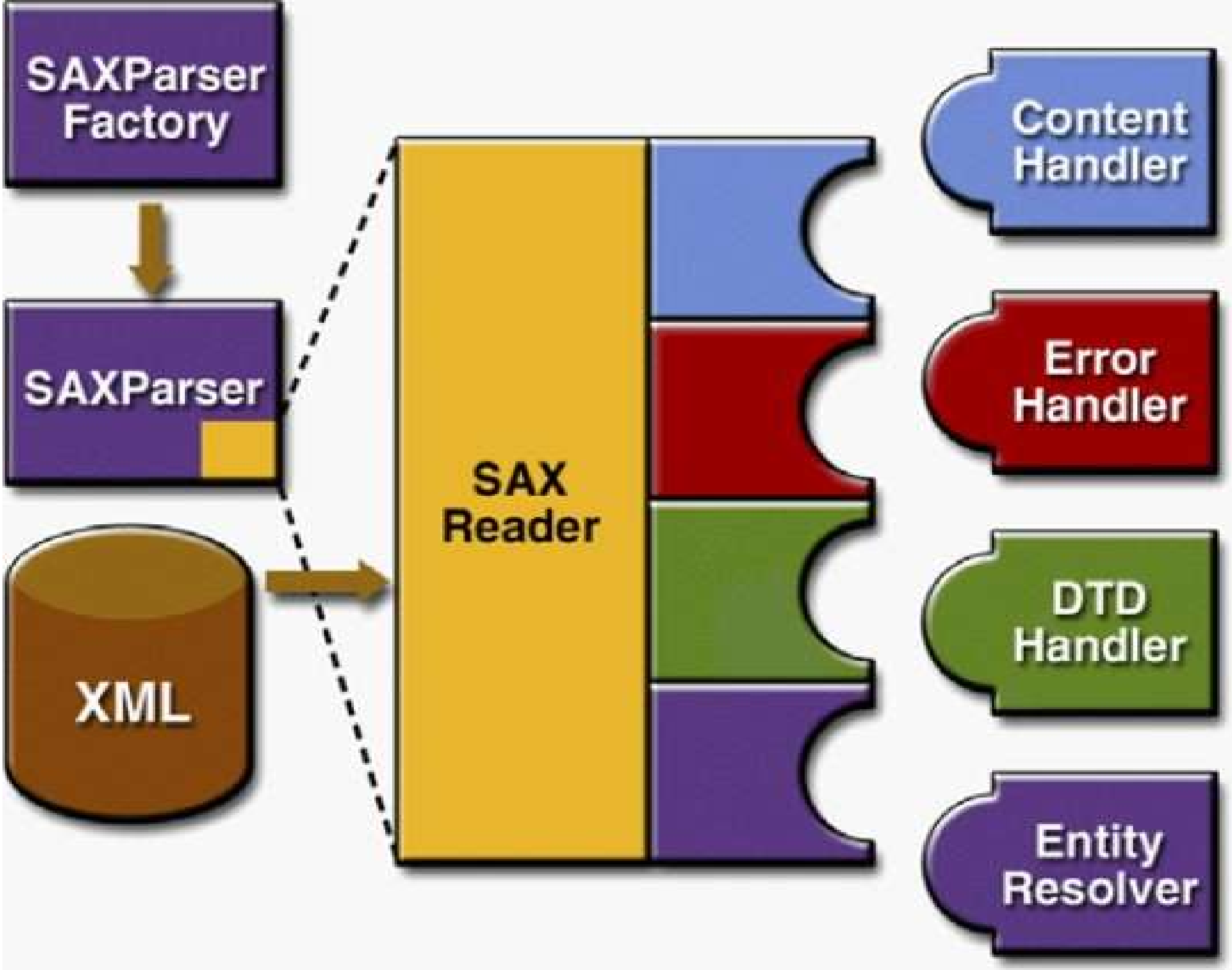
```
Zaczynamy...
Początek elementu: autor
Ciąg znaków:    \n
Początek elementu: imię
Ciąg znaków:    Paweł
Koniec elementu:  imię
Ciąg znaków:    \n
Początek elementu: nazwisko
Ciąg znaków:    Nieznany
Koniec elementu:  nazwisko
Ciąg znaków:    \n
Początek elementu: e-mail
Ciąg znaków:    noname@wp.pl
Koniec elementu:  e-mail
Ciąg znaków:    \n
Koniec elementu:  autor
Koniec.
```

SAX – przykład (2/3)

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;

class SAXHandler extends DefaultHandler  {
    public void startDocument() {   System.out.println("Zaczynamy..."); }

    public void endDocument() { System.out.println("Koniec."); }

    public void startElement(String namespaceURI, String localName, String rawName, Attributes atts)
    { System.out.println("Początek elementu: " + localName); }

    public void endElement(String namespaceURI, String localName, String rawName) {
        System.out.println("Koniec elementu:  " + localName); }

    public void characters(char ch[], int start, int length) {
              System.out.print("Ciąg znaków:    ");
              for (int i = start; i < start + length; i++) {
                switch(ch[i]) {
                case '\n':
                  System.out.print("\\n");
                  break;
                case '\t':
                  System.out.print("\\t");
                  break;
                default:
                  System.out.print(ch[i]);
                  break;
                }
              }
              System.out.println();
            }
```

SAX – przykład (3/3)

```
    public void error(SAXParseException ex) throws SAXException  {
        System.err.println("Błąd (" + ex.getSystemId() + ":"
                + ex.getLineNumber() + ":" + ex.getColumnNumber()
                + ") " + ex.getMessage());
        throw ex;
    }
// metody fatalerror(...) oraz warning(...)


  public class SAXExample
  {
       public static void main(String[] args) throws Exception
       {
         String uri;
         if (args.length == 0){
             throw new Exception("Podaj nazwę pliku");
         } else {
             uri = args[0];
           }
         XMLReader parser = XMLReaderFactory.createXMLReader();
         SAXHandler handler = new SAXHandler();
         parser.setContentHandler(handler);
         parser.setErrorHandler(handler);
         parser.parse(uri);
       }
   }
```


SAXParser Factory
SAXParser
XML
SAX Reader
Content Handler
Error Handler
DTD Handler
Entity Resolver

DOM

- Widząc różne dokumenty XML łatwo można zauważyć, że posiadają one strukturę drzewiastą:
  - Istnieje tylko jeden element główny, który można traktować jako korzeń drzewa
  - Element może zawierać w sobie inne elementy
- **DOM** (Document Object Model) wykorzystuje ten naturalny sposób reprezentacji dokumentu
- Rekomendacja W3 Consortium
- org.w3c.dom
- Niezależny od systemu operacyjnego oraz języka programowania (dzięki IDL)
- Cele:
  - Dać łatwy dostęp do elementów dokumentu
  - Umożliwić dodawanie, usuwanie i edycję elementów oraz ich atrybutów
- Po sparsowaniu dokument przechowywany jest cały czas w pamięci w postaci drzewa powiązanych ze sobą obiektów
- Wszystko jest węzłem (Node)

Interface Definition Language

- Notacja zdefiniowana przez Object Management Group (http://www.omg.org)
- IDL

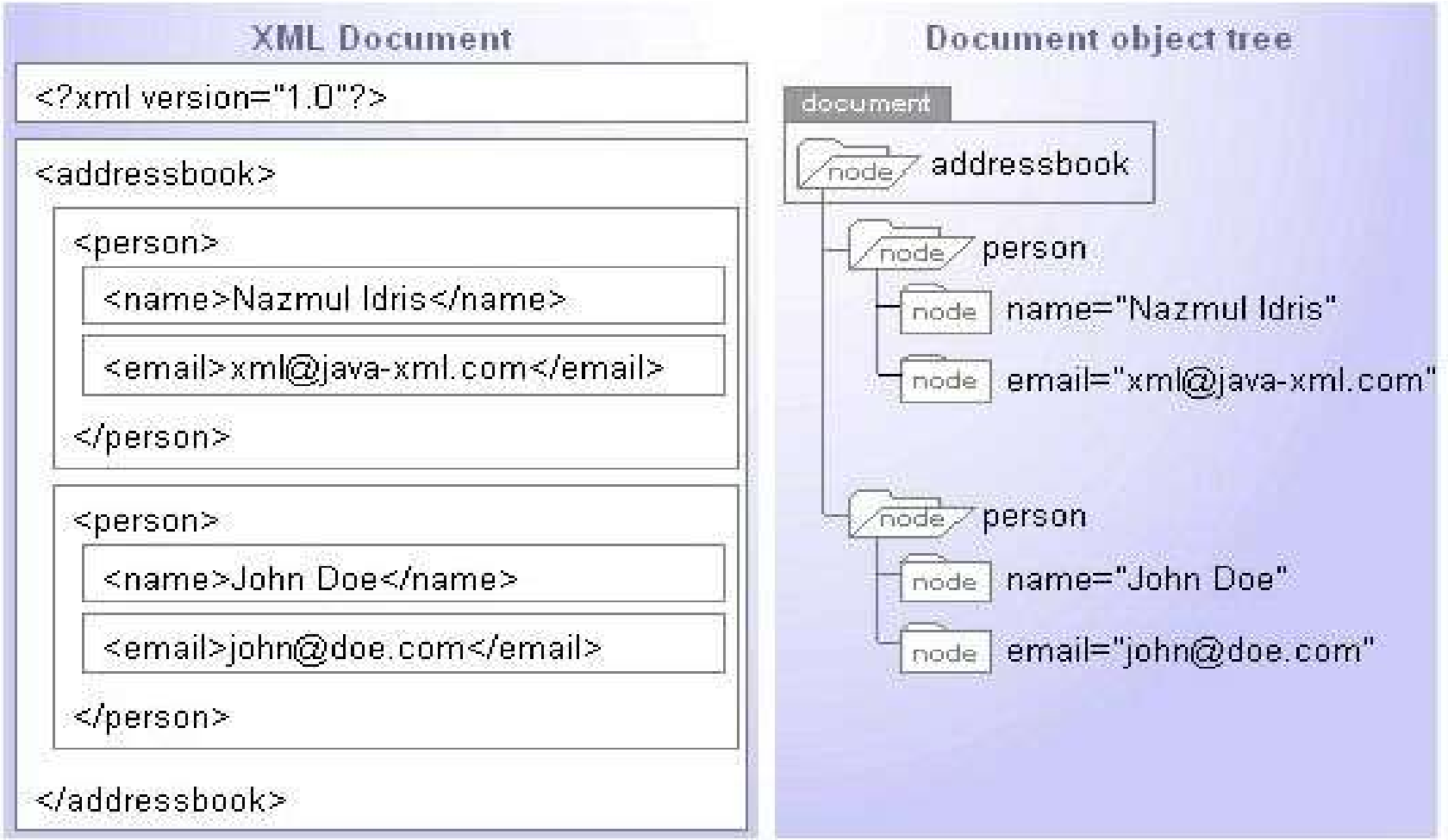
```
interface Attr : Node {
  readonly attribute DOMString name;
  readonly attribute boolean   specified;
           attribute DOMString value;
  readonly attribute Element   ownerElement;
};
```

- Java

```
public interface Attr extends Node {
  public String getName();
  public boolean getSpecified();
  public String getValue();
  public void setValue(String value) throws DOMException;
  public Element getOwnerElement();
}
```

DOM - przykłady

1) **<email>deekay@gazeta.pl </email>** - jeden węzeł, nazwa taga=email, typ wezła to Node.ELEMENT_NODE
2)



Historia DOM

- Netscape wymyśla język JavaScript i umieszcza do w swojej przeglądarce (jej DOM dawał pewien ograniczony dostęp do elementów HTML, jego możliwości zostały rozszerzone dopiero w Netscape 3)
- Internet Explorer (Microsoft) – alternatywa dla przeglądarki Netscape
- IE 3 kopiował DOM z Netscape jednak czynił to w niepełny sposób (pierwsze problemy ze zgodnością)
- DOM Level 0 - do dziś obsługiwany przez przeglądarki gdyż w sieci jest nadal wiele stron używających tej “wersji” DOM
- Netscape 3 – wzorzec dla wszystkich jeśli chodzi o DOM Level 0 (do którego trzeba się dostosować)
- Przeglądarki w wersji 4 dawały możliwość obsługi DHTML, DOM musiał być więc udoskonalony
- Od tego momentu Microsoft postanawia nie kopiować rozwiązań Netscape (rezultat: kompletny brak zgodności)
- The Browser Wars (konkurencji walczą, programiści webowi płaczą)
- W chwili obecnej DOM Microsoftu jest uważany za lepszy i bliższy standardowi stworzonemu przez W3C

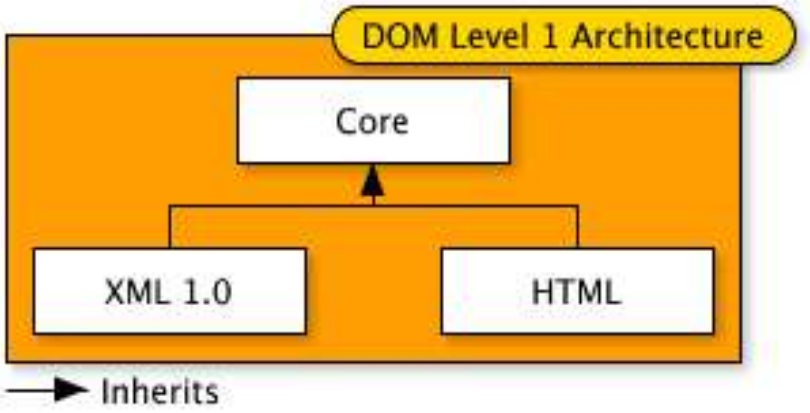
Ewolucja DOM

- W3C Activity – część W3C odpowiedzialna za DOM (od sierpnia 1997)
- DOM Level 0 - nie jest specyfikacją
- **DOM Level 1** - (październik 1998), dostarczał wsparcie dla XML 1.0 oraz HTML 4.0 (osiąga pewien stopień zgodności ze wszystkimi przeglądarkami)
- **DOM Level 2** - (listopad 2000), dostarczał wsparcie dla przestrzeni nazw XML, wsparcie dla Cascading Style Sheets (CSS) oraz mechanizmy do poruszania się po drzewie. W styczniu 2003 stał się rekomendacją W3C
- **DOM Level 3** - (kwiecień 2004), przede wszystkim dodał mechanizm ładowania i zapisywania dokumentu

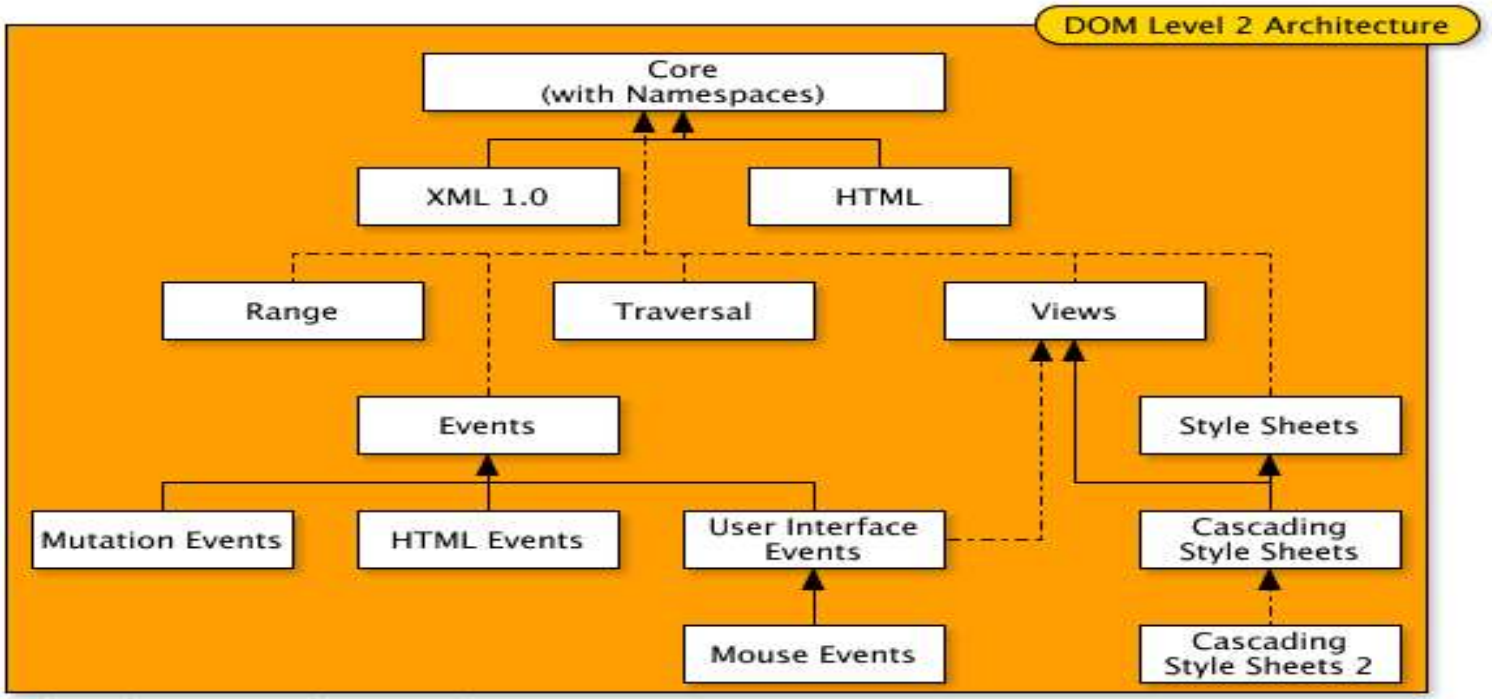
- Inne moduły:
  - **DOM for MathML 2.0**
  - **DOM for SMIL Animation** - (Synchronized Multimedia Integration Language)
  - **DOM for SVG 1.0** - (Scalable Vector Graphics) DOM dla języka do opisywania grafiki dwuwymiarowej
- **DOM Test Suites**

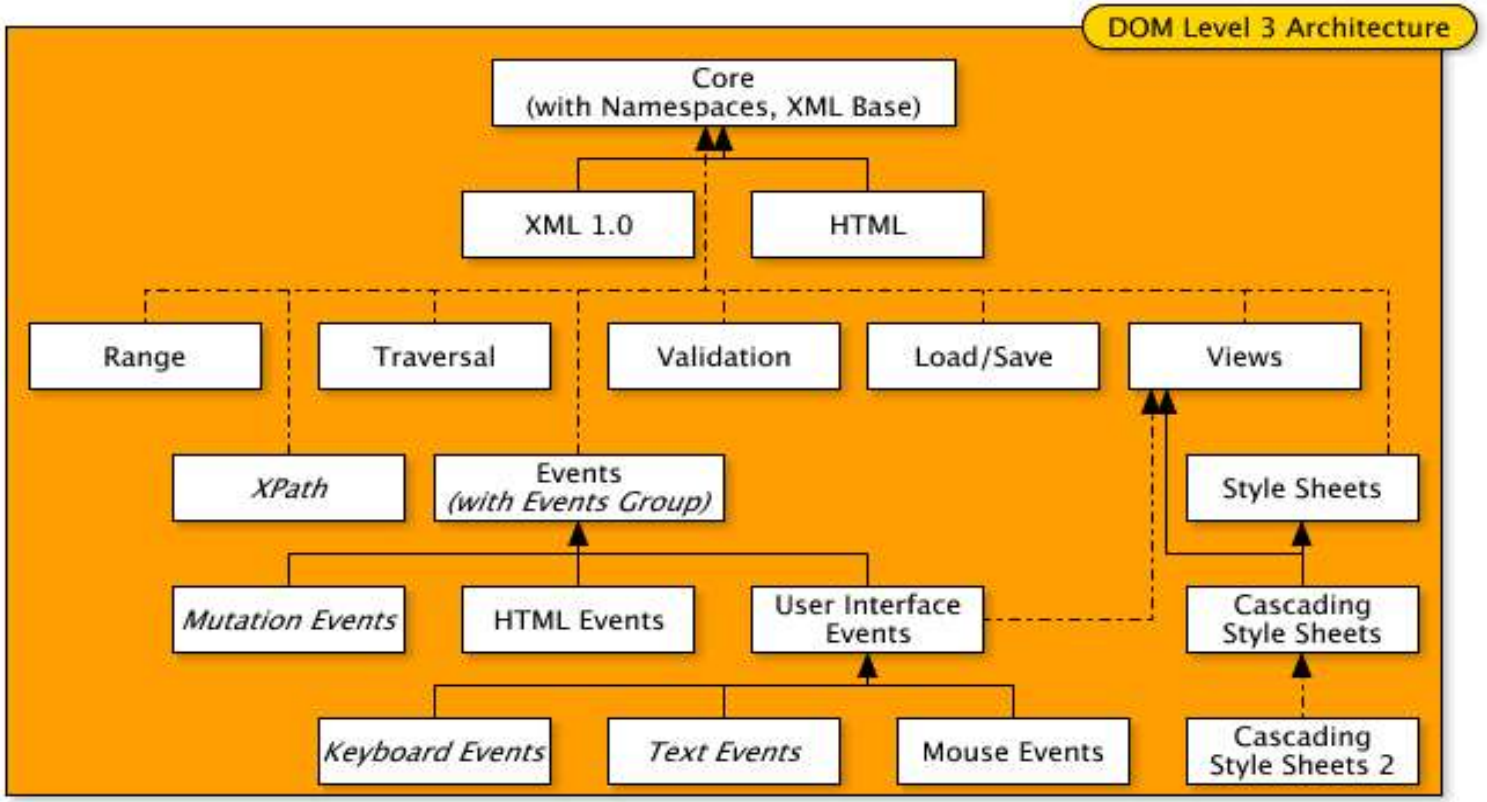
Architektura DOM

Podział na moduły



- **Core**
- **HTML**
- **Cascading Style Sheets**
- **Events -** daje aplikacjom możliwość reakcji na zdarzenia występujące podczas wizualizacji i oglądania dokumentu (mouse events, keyboard events, zdarzenia specyficzne dla HTML)
- **Views –** metody umożliwiające prezentację graficzną dokumentu po zastosowaniu stylów CSS
- **Traversal & Range** - pozwalający na wygodne przemieszczanie się po drzewie dokumentu i wybieranie jego fragmentów
- **Validation –** zbiór metod do modyfikacji drzewa tak aby nadal było ono `valid`
- **Load & Save**
- **XPath**


DOM Level 2 Architecture
Core
(with Namespaces)
XML 1.0
HTML
Range
Traversal
Views
Events
Style Sheets
Mutation Events
HTML Events
User Interface
Events
Cascading
Style Sheets
Mouse Events
Cascading
Style Sheets 2


DOM Level 3 Architecture
Core
(with Namespaces, XML Base)
XML 1.0
HTML
Range
Traversal
Validation
Load/Save
Views
XPath
Events
(with Events Group)
Style Sheets
Mutation Events
HTML Events
User Interface
Events
Cascading
Style Sheets
Keyboard Events
Text Events
Mouse Events
Cascading
Style Sheets 2

Metody dostępu do drzewa

Chcąc wykonać jakiekolwiek działania na drzewie DOM mamy dostępne w tym celu dwie równoważne metody działania zawarte w module DOM Core:

- poprzez generyczny interfejs Node, który zawiera minimalny zbiór atrybutów i metod potrzebnych do pracy z drzewem oraz wskaźniki umożliwiające zlokalizowanie ojca, rodzeństwa i dzieci

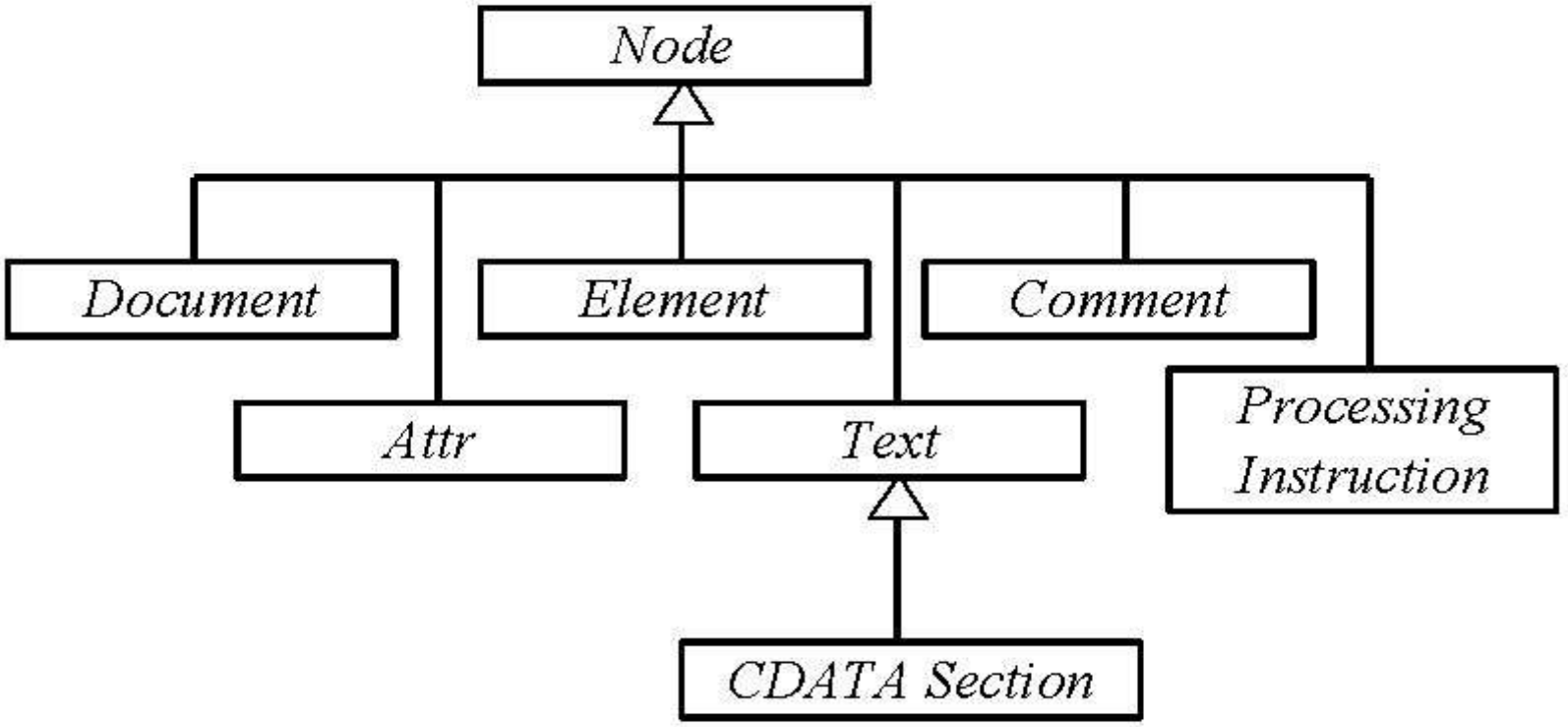
- poprzez specyficzny interfejs dla każdego typu (zbiór interfejsów dla każdej możliwej części dokumentu – każdy dziedziczy z Node może więc oprócz swoich metod korzystać również z tych odziedziczonych)

Interfejs Node

- Atrybuty (DOCUMENT_NODE, ELEMENT_NODE, COMMENT_NODE, ...)
- Metody
  - getNodeName()
  - getNodeValue()
  - getNodeType()
  - getAttributes()
  - getChildNodes()
  - getFirstChild()
  - getLastChild()
  - getNextSibling()
  - getPreviousSibling()
  - getParentNode()
  - hasChildNodes()
  - appendChild(Node)
  - insertBefore(Node, Node)
  - removeChild(Node)
  - setNodeValue(String)
  - setNodeName(String)
  - ...

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	`"#cdata-section"`	content of the CDATA Section	null
Comment	`"#comment"`	content of the comment	null
Document	`"#document"`	null	null
DocumentFragment	`"#document-fragment"`	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	`"#text"`	content of the text node	null

DOM Core



- CDATASection (brak własnych metod i atrybutów)
- Comment (brak własnych metod i atrybutów, nie jest wymagane aby parser tworzył te obiekty)

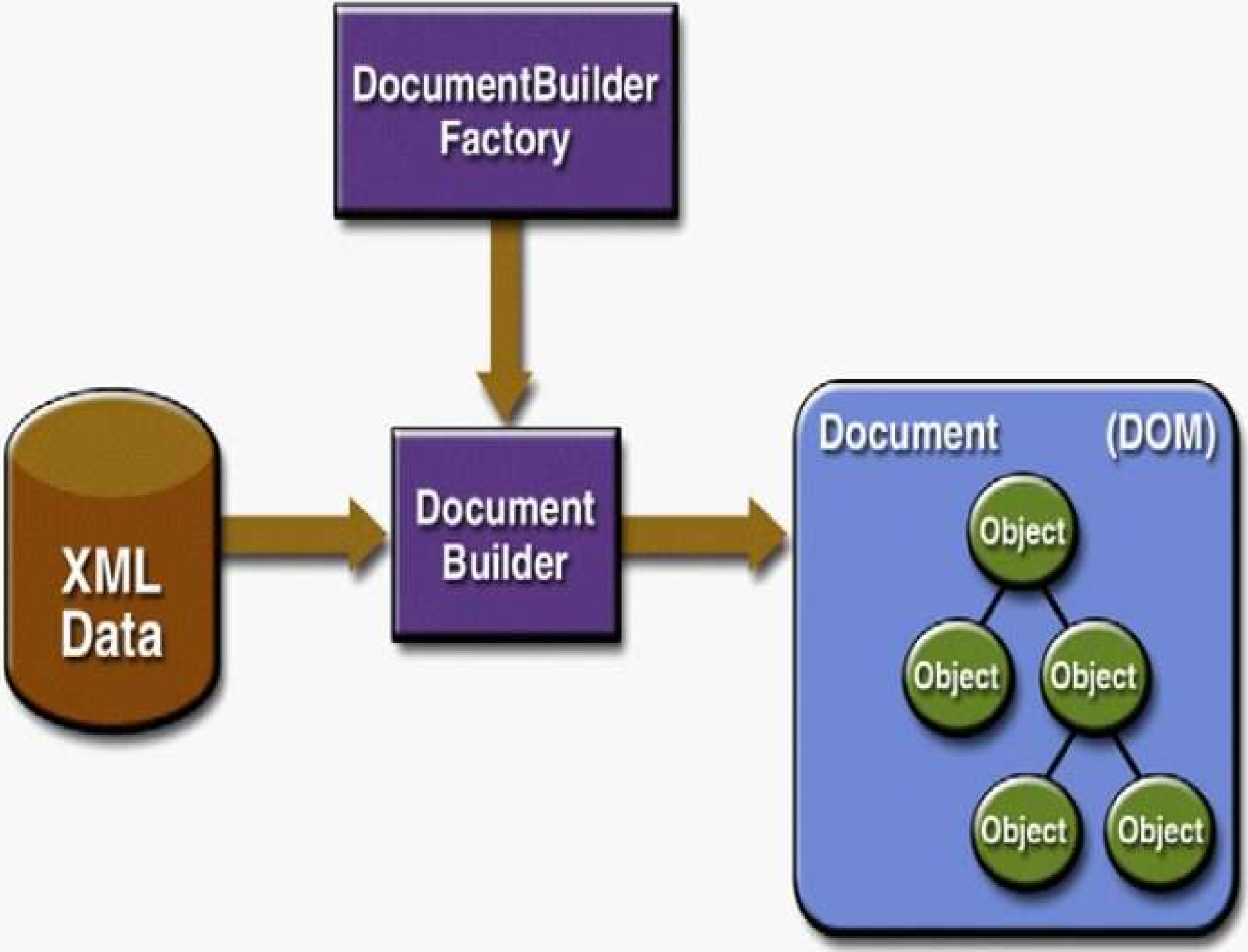
DOM Core

- Document
  - reprezentuje dokument
  - createX(...)
  - węzły związane z jedną instancją obiektu Document, mogą być wstawione jedynie do tego obiektu
  - importNode() - DOM Level 2
- DocumentType (tylko jeden dla danego dokumentu)
- DocumentFragment
- Text
  - początkowo cały nieoddzielony niczym tekst jest w jednym węźle
  - splitText() - rozdziela tekst, gdy np. chcemy coś wstawić pomiędzy
- DOMImplementation – posiada metodę hasFeature(...)

- NodeList (zastosowanie: lista dzieci)
- NamedNodeMap (zastosowanie: dostawanie się do atrybutów)

SimpleDOM.java

```
public class SimpleDOM {

 public static void main(String[] args) {

    String fileName = args[0];
    DocumentBuilderFactory factory =
                      DocumentBuilderFactory.newInstance();
    factory.setValidating(false);
     DocumentBuilder builder = factory.newDocumentBuilder();
     Document document = builder.parse(fileName);

    traverse(document);

     private static void traverse(Node node) {
        int nodeType = node.getNodeType();
        if (nodeType == Node.TEXT_NODE) {
           System.out.println("TXT: ["+node.getNodeValue()+"]");
        } else
             if (nodeType == Node.ELEMENT_NODE || nodeType ==
                                              Node.DOCUMENT_NODE) {
                NodeList chldrn = node.getChildNodes();
                for (int i = 0; i < chldrn.getLength(); i++) {
                     traverse(chldrn.item(i));
                }
             }
     }
 }
}
```


DocumentBuilder Factory
XML Data
Document Builder
Document (DOM)
Object
Object
Object
Object
Object

DOM vs. SAX

- **DOM**
  - Stworzony przez konsorcjum W3C
  - Swobodne chodzenie po drzewie
  - Ma problem z dużymi dokumentami (cały dokument trzymany w pamięci, duży narzut w stosunku do rozmiaru samego dokumentu)
  - Kosztowny czasowo (budowanie drzewa)
  - Pozwala tworzyć i modyfikować istniejące dokumenty

- **SAX**
  - Stworzony przez osoby z grupy dyskusyjnej
  - Dostęp sekwencyjny (nie możemy wrócić)
  - Działa bezproblemowo na dowolnie dużych dokumentach
  - Nie ponosi zbyt dużych kosztów czasowych
  - Dokument można tylko czytać

DOM Test Suites (DOM TS)

- Stworzone przez W3C DOM Activity
- http://www.w3.org/DOM/Test/ - FAQ, opisy, przykładowe pliki
- Zawiera testy dla każdego poziomu specyfikacji DOM (1, 2, 3)
- Testy te reprezentowane są w gramatyce XML (w formie DTD oraz XML Schema), mogą być przeniesione w formatu opisowego do formatu zrozumiałego przez dany język programowania

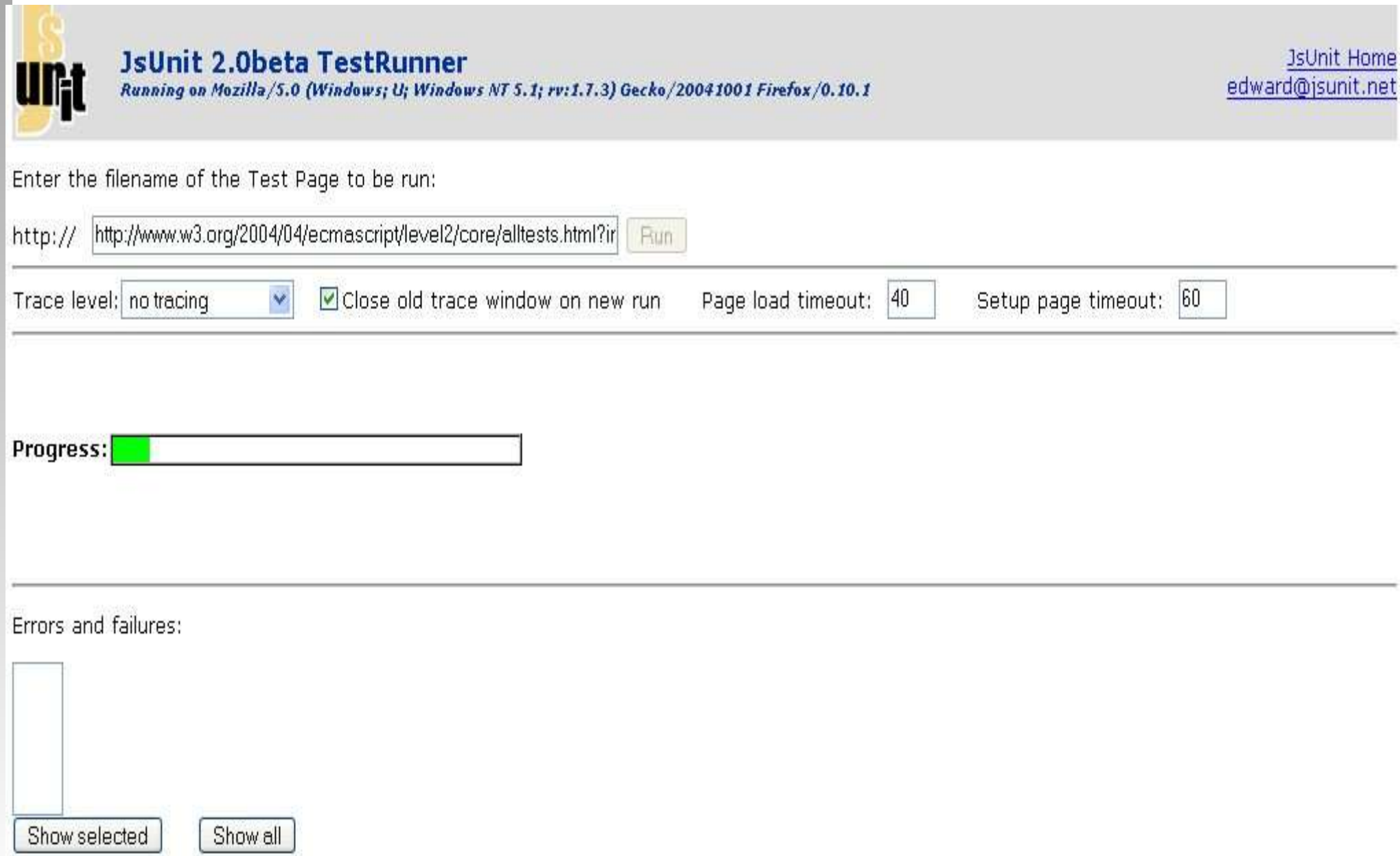
DOM Module	Java (download)	ECMAScript (run the tests)
Level 1 Core	20040405	20040405
Level 2 Core	20040405	20040405
Level 2 HTML	20040405	20040405
Level 3 Core	20040405	
Level 3 Load and Save	20040405	20040405
Level 3 Validation	20040405	N/A

DOM TS - Java

- `java -jar <dom-test.jar>`

```
Implementation title:Java Runtime Environment
Implementation vendor:Sun Microsystems, Inc.
Implementation version:1.4.2_04
Parser supports DOM Level 1 XML
Parser does not support DOM Level 1 HTML
Parser supports DOM Level 2
Parser supports DOM Level 2 XML
Parser does not support DOM Level 2 HTML
Parser does not support DOM Level 2 Views
Parser does not support DOM Level 2 Style Sheets
Parser does not support DOM Level 2 CSS
Parser does not support DOM Level 2 CSS2
Parser does not support DOM Level 2 Events
Parser does not support DOM Level 2 User Interface Events
Parser does not support DOM Level 2 Mouse Events
Parser does not support DOM Level 2 Mutation Events
Parser does not support DOM Level 2 HTML Events
Parser does not support DOM Level 2 Traversal
Parser does not support DOM Level 2 Range
Parser does not support DOM Level 3
Parser does not support DOM Level 3 XML
Parser does not support DOM Level 3 Core
Parser does not support DOM Level 3 Events
Parser does not support DOM Level 3 XPath
Parser does not support DOM Level 3 Load & Save
Parser does not support DOM Level 3 Asynchronous Load & Save
Parser does not support DOM Level 3 Validation
Parser does not support SVG Version 1.0
Parser does not support SVG Version 1.0 Static
Parser does not support SVG Version 1.0 Dynamic
Parser does not support SVG Version 1.0 Animation
Parser does not support SVG Version 1.0 <full support>
Parser does not support SMIL Animation
Parser does not support MathML Version 2.0


hasNullString=true
signed=true
coalescing=false
expandEntityReferences=false
ignoringElementContentWhitespace=false
namespaceAware=false
validating=false
```

DOM TS - ECMAScript



JDOM – przyczyny powstania

DOM pomimo wielu swoich zalet zawiera też wiele ograniczeń:

- Powinien być, w stopniu w jakim jest to możliwe, zgodny wstecz ze starymi, kiepsko przemyślanymi, modelami obiektowymi z przeglądarek trzeciej generacji
- Zaprojektowany przez komitet aby pogodzić różnice między modelami implementowanymi przez Microsoft, Netscape i innych producentów. W3C zaproponowało rozwiązanie akceptowalne w choćby w minimalnym stopniu przez wszystkich.
- DOM jest API niezależnym od języka. Jest ograniczony do cech, które będą dostępne we wszystkich językach do implementujących (również w tych nie w pełni obiektowych jak JavaScript, Visual Basic)
- Musi pracować zarówno z HTMLem (w wersji XHTML i tradycyjnej) jak i z XMLem

JDOM

- http://www.jdom.org - źródła. przykłady, dokumentacja
- Open source (jest dostępny na serwerze CVS z dostępem publicznym)
- Data stworzenia: jesień 2000
- Twórcy: Brett McLaughlin, Jason Hunter
- Obecnie dostępny w wersji 1.0 w postaci pliku jar (150 kB)
- Zaprojektowany tylko dla Javy i tylko dla XMLa
- ~(“Java Document Object Model”)
- Znacznie prostszy i “czystszy” (nie ma już konieczności utrzymania zgodności wstecz)
- Bardziej intuicyjny
- Napisanie tego samego programu w JDOM i W3C DOM najczęściej zajmie mniej czasu i po drodze napotka się na mniej błędów
- JDOM - DOM (RMI – CORBA. Java – C++)
- Java Specification Request JSR-102

JDOM

- Zasada 80-20
- Reprezentacja drzewiasta dokumentu
- Nie posiada żadnych pomocniczych klas do przechodzenia drzewa
- Jest zestawem konkretnych klas (reprezentujących różne rodzaje wezłów) a nie interfejsów
- Podstawowe klasy: Document, Element, Attribute, Text, Comment, ProcessingInstruction, Namespace, DocType, CDATA, EntityRef
- Węzły mogą należeć do obiektu Document lub istnieć niezależnie
- JDOM korzysta z bibliotek Javy i z konwencji tworzenia kodu w Javie
- Wszystkie klasy mają metody: `equals(), hashCode(), toString()`
- Wszystkie implementując interfejsy: `Cloneable` (z wyjątkiem Namespace) i `Serializable`. Klon elementu nie ma ojca i może być dodany zarówno do tego samego dokumentu jak i do innego
- Dzieci klasy Element przechowywane są w klasie java.util.List
- Nie posiada parsera (wykorzystuje SAX do parsowania dokumentu i budowy drzewa)

JDOM

- Potrafi konwertować obiekt DOM Document na JDOM Document...
- ..może też zbudować drzewo od podstaw (tak jak DOM) oraz z bazy danych
- Nie pozwala stworzyć komentarzy, których dane zawierają ciąg znaków “--” oraz elementów i atrybutów, które w swojej przestrzeni nazw są w konflikcie
- Można modyfikować drzewo (tak jak w DOM) z zachowaniem pewnych restrykcji (np. atrybut można dodać do elementu ale nie do komentarza)
- Synchronizacja obiektów spada na programistę, JDOM nie gwarantuje bezpiecznego dostępu do swoich klas przez wiele wątków
- Metoda `equals()` sprawdza indentyczność obiektów (elem1.equals(elem2) tylko jeśli elem1 == elem2). Powodem jest fakt, że w dokumencie XML porządek elementów oraz ich pozycja ma znaczenie

```
public final boolean equals(Object o) {
  return (this == o);
}
```

JDOM – toString() oraz hashCode()

toString()

```
[Document:  No DOCTYPE declaration, Root is [Element: <html
 [Namespace: http://www.w3.org/1999/xhtml]/>]]
[Element: <html [Namespace:http://www.w3.org/1999/xhtml]/>]
[Attribute: xml:lang="en"]
[Text:
]
[Attribute: type="text/css"]
[Attribute: rel="stylesheet"]
[Text: Latest Version: ]
[Element: <a [Namespace: http://www.w3.org/1999/xhtml]/>]
[Attribute: href="http://www.rddl.org/"]
[Text: June 16, 2002]
```

hashCode()

```
public final int hashCode() {
   return super.hashCode();
 }
```

Pakiety JDOM

- org.jdom - (Element, Namespace, Text, Comment, ...)
- org.jdom.input – klasy do budowania drzewa JDOM (SAXBuilder, DOMBuilder, ..)
- org.jdom.output (XMLOutputter, SAXOutputter, DOMOutputter, ...)
- org.jdom.xpath – jedyna klasa XPath umożliwia przechodzenie drzewa za pomocą wywołań XPath

Oraz kilka innych w tym dodatkowe dodające takie klasy jak:

- JTreeOutputter
- ResultSetBuilder

Tworzenie dokumentu

- <root/>

```
Document doc = new Document(new Element("root"));
```

-
```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(url);
```

- <root>This is the root</root>

```
Document doc = new Document();
Element e = new Element("root");
e.setText("This is the root");
doc.addContent(e);

//Document doc = new Document(

//new Element("root").setText("This is the root"));
```

- DOM

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element root = doc.createElement("root");
Text text = doc.createText("This is the root");
root.appendChild(text);
doc.appendChild(root);
```

XMLOutputter

- ```
  XMLOutputter outp = new XMLOutputter();
  outp.output(doc, fileStream);

  //outp.output(doc, socketStream);

  //outp.output(doc, System.out);
  ```
- Klasa Format – zawiera szereg metod do formatowania wyjścia
- ```
  XMLOutputter outp = new XMLOutputter
                          (Format.getPrettyFormat());
  ```

Poruszanie się po drzewie

- Pobranie głównego elementu

```
Element root = doc.getRootElement();
```

- Pobranie listy dzieci

```
List allChildren = root.getChildren();
```

- Pobranie tylko elementów o określonej nazwie...

```
List namedChildren = root.getChildren("name");
```

- ...oraz pierwszego elementu o określonej nazwie

```
Element child = root.getChild("name");
```

- Lista zwracana przez metodę `getChildren(...)` jest listą "żywą" - jakakolwiek zmiana dokonana na liście natychmiast ma wpływ na zmianę w drzewie
- Usuwanie czwartego dziecka

```
allChildren.remove(3);
```

- Usunięcie dziecka o imieniu "jack"

```
allChildren.removeAll(root.getChildren("jack"));
```

Praca z dokumentem w trybie mieszanym

```
<root>
  <!-- Komentarz -->
   Kilka zdań na bardzo ciekawe tematy.
  <elem>Tekst przyporządkowany elementowi elem</elem>
</root>
```

- Obsługa tryby mieszanego w JDOM jest bardzo prosta

```
String text = root.getTextTrim();  // "Kilka zdań ..."
Element elem = root.getChild("elem");
```

- W bardziej skomplikowanych przypadkach korzystamy z listy

```
List mixedCo = root.getContent();
Iterator itr = mixedCo.iterator();
while (itr.hasNext()) {
  Object o = i.next();
  if (o instanceof Comment /* Element, Text*/) {
    ...
  }
}
```

Przestrzenie nazw

- Tworzenie nowej przestrzeni nazw

```
Namespace xhtml = Namespace.getNamespace(
  "xhtml", "http://www.w3.org/1999/xhtml");
```

- Praca z przestrzeniami nazw

```
elt.addContent(new Element("table", xhtml));

List kids = html.getChildren("title", xhtml);
```

- Jeśli żadna przestrzeń nazw nie jest podana element jest tworzony bez przestrzeni nazw
- Przestrzeń nazw elementu jest na stałe z nim związana. JDOM czuwa nad tym aby nie zmieniała się ona wskutek "poruszania" się elementu
- Jeśli element nie miał przyporządkowanej sobie przestrzeni nazw i zostanie przesunięty "pod" element, który posiada przestrzeń nazw, to nie odziedziczy jej po ojcu

ResultSetBuilder (1/2)

- Rozszerzenie stworzone dla ludzi, którzy chcieliby traktować rezultat zapytania SQL jako dokument XML
- org.jdom.contrib.input
- Kod

```
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery("select id, name from regi-
    stry");
ResultSetBuilder builder = new ResultSetBuilder(rs);
Document doc = builder.build();
```

- Rezultat

```
<result>
  <entry>
    <id>1</id>
    <name>Alice</name>
  </entry>
  <entry>
    <id>2</id>
    <name>Bob</name>
  </entry>
</result>
```

ResultSetBuilder (2/2)

- ResultSetMetaData – klasa pozwalająca na zmianę nazw głównego elementu (`setRootName(Stringname)`), kolumn (`setRowName(Stringname)`) lub przypisać przestrzeń nazw (`setNamespace(Namespaces)`)
- Możemy również kolumny reprezentować w pliku XML jako atrybuty
- Kod

```
ResultSetBuilder builder = new ResultSetBuilder(rs);
builder.setAsAttribute("id");
builder.setAsElement("name", "fname");
//dla atrybutów setAsAttribute (StringcolumnName, Stringattrib-
   Name)
Document doc = builder.build();
```

- Rezultat

```
<result>
  <entry id="1">
    <fname>Alice</fname>
  </entry>
  <entry id="2">
    <fname>Bob</fname>
  </entry>
</result>
```

Testy wydajnościowe

- "A look at features and performance of XML document models in Java" Dennis Sosnoski
- porównanie rożnych bibliotek i API, które potrafią pracować z dokumentami XML

Porównywane modele:

- DOM (dwie implementacje: Xerces, Crimson)
- JDOM
- dom4j
- Electric XML
- XML Pull Parser

Xerces, Crimson

Xerces

- http://xml.apache.org/xerces-j
- Open-source (dostępny jako jar)
- Bazuje na XML4J – parserze IBM (Java)
- Obsługa SAX2, DOM (brak wsparcia dla łączenia tych API)
- Wsparcie dla walidacji plików (DTD, XML Schema)
- Specjalny tryb: Xerces deferred (dokument na początku jest reprezentowany w bardziej "ścisłej" formie i jest rozwijany do "pełnego" drzewa DOM tylko gdy zachodzi taka potrzeba)
- Xerces deferred = szybsze parsowanie + mniejsze zużycie pamięci

Crimson

- http://xml.apache.org/crimson
- Open-source (znacznie mniejszy plik jar niż w przypadku Xerces)
- Bazuje na parserze – Sun Project X
- Waliduje pliki (wsparcie dla DTD)
- Obsługa SAX2, DOM
- Tworzenie drzewa DOM (poprzez parser SAX2)

dom4j

dom4j

- http://dom4j.org
- Projekt, który odłączył się w 2000 r. (James Strachan, open-source)
- Java-only
- Cele: łatwość użycia i intuicyjność (jak w JDOM)
- Do reprezentacji wezłów użyto interfejsów budowanych przez metody fabryki zamiast konkretnych klas budowanych poprzez konstruktory
- Generyczny interfejs Node, wsparcie dla XPath, XML Schema
- Umożliwia parsowanie oparte na zdarzeniach dla bardzo dużych dokumentów
- Zdaniem niektórych łatwiejszy w pracy niż JDOM
- Daje wiekszą elastyczność w programowaniu niż JDOM kosztem bardziej złożonego API
- Tworzonym klasą można narzucać ograniczenia (np. własność nazwa z Element musi być poprawną nazwą XMLową)

Electric XML

Electric XML

- http://www.themindelectric.com/exml/
- Projekt komercyjny (Java)
- Drzewiasta reprezentacja dokumentu
- Bardzo mały plik jar (50kb ok. 10% rozmiaru konkurenta – dom4j)
- Bardzo proste API
- Wsparcie dla XPath
- Brak walidacji dokumentów
- Reprezentacja dokumentu budowana jedynie z pliku tekstowego
- Działa jedynie z ograniczonym podzbiorem plików XML
- Podobne podejście jak w JDOM (klasy zamiast interfejsów)
- Nie używa klas kolekcji (w przeciwieństwie do JDOM)
- Usuwa białe znaki jeśli nie są częścią tekstu
- Działa dobrze dla aplikacji gdzie białe znaki są bez znaczenia...
- ...gorzej gdy są istotne (aplikacje, które mają wyświetlić dokument w przeglądarce)
- Ograniczenie to może mieć wpływ na wyniki testów

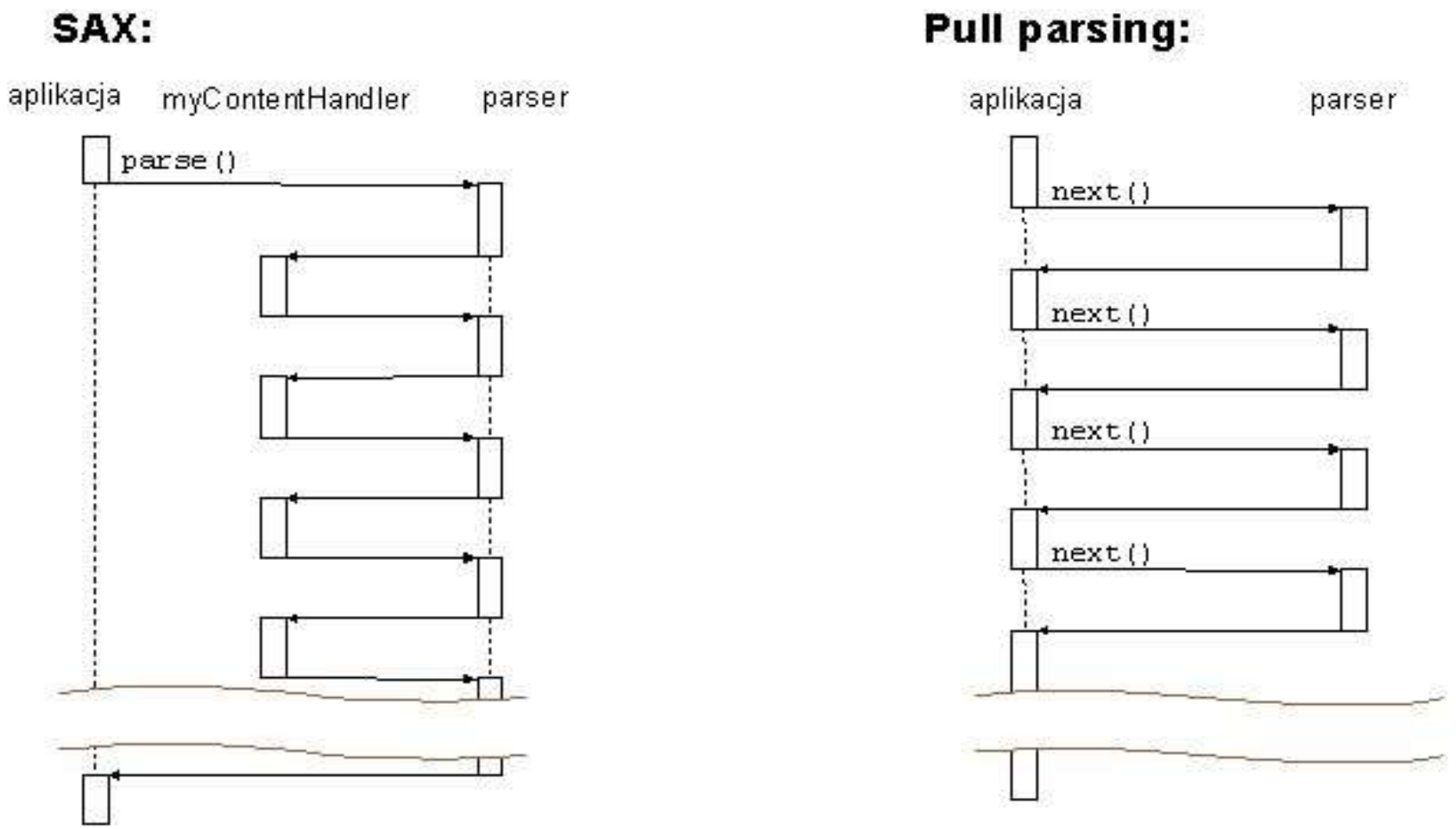
XML Pull Parser

- http://www.extreme.indiana.edu/xgws/xsoap/xpp/
- Prezentuje inne podejście do parsowania XML
- Najprostsze API z testowanych
- Brak walidacji
- Mała liczba klas, nie używa klas kolekcji, działa jedynie z podzbiorem plików XML, buduje reprezentacje dokumentu jedynie z plików tekstowych, bardzo mały plik jar (jak EXML)
- Brak wsparcia dla komentarzy, encji, instrukcji przetwarzania
- Struktura dokumentu złożona jedynie z elementów, atrybutów, przestrzeni nazw i tekstu
- Zawiera dodatkowy tryb: pull-parser
- W trybie tym parsowanie dokumentu jest odkładane dopóki nie ma żądania dostępu do części dokumentu

Przetwarzanie strumieniowe – pull parsing

- Alternatywa dla modelu zdarzeniowego
- Przetwarzanie kontrolowane przez aplikację a nie parser
- Aplikacja “wyciąga” kolejne zdarzenia z parsera
- Parser działa podobnie jak iterator czy kursor
- Zachowane cechy modelu SAX
  - Duża wydajność
  - Możliwość przetwarzania dowolnie dużych dokumentów
- Korzyści:
  - Możliwość przerwania przetwarzania przed końcem pliku, gdy potrzebujemy z niego tylko część danych

Pull parsing a SAX



Testowane pliki

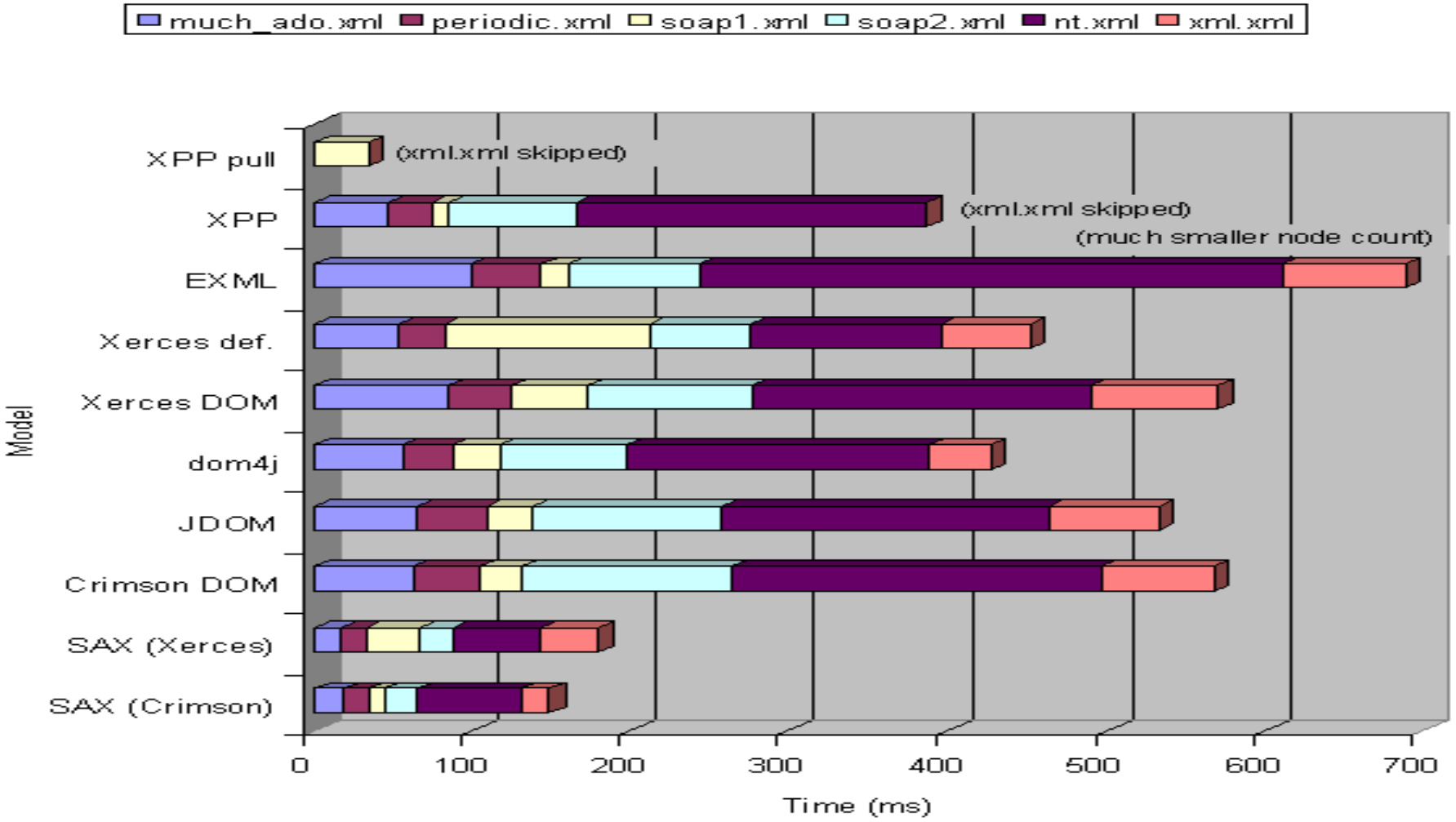
Testy przeprowadzene były na kilku plikach XML mających pokazać różne możliwe użycie XMLa:

- much_ado.xml - sztuka Szekspira w XML, bez atrybutów (200 kb)
- periodic.xml - układ okresowy pierwiastków w XML, niewiele atrybutów (117 kb)
- soap1.xml (0.4 kb)
- soap2.xml (134 kb)
- nt.xml - Nowy Testament, brak atrybutów, dużo tekstu (1 MB)
- xml.xml - specyfikacja XML w XML, styl mieszany, występują atrybuty (160 kb)

W przypadku soap1.xml jeden test oznacza powtórzenie 49 razy tego samego testu.

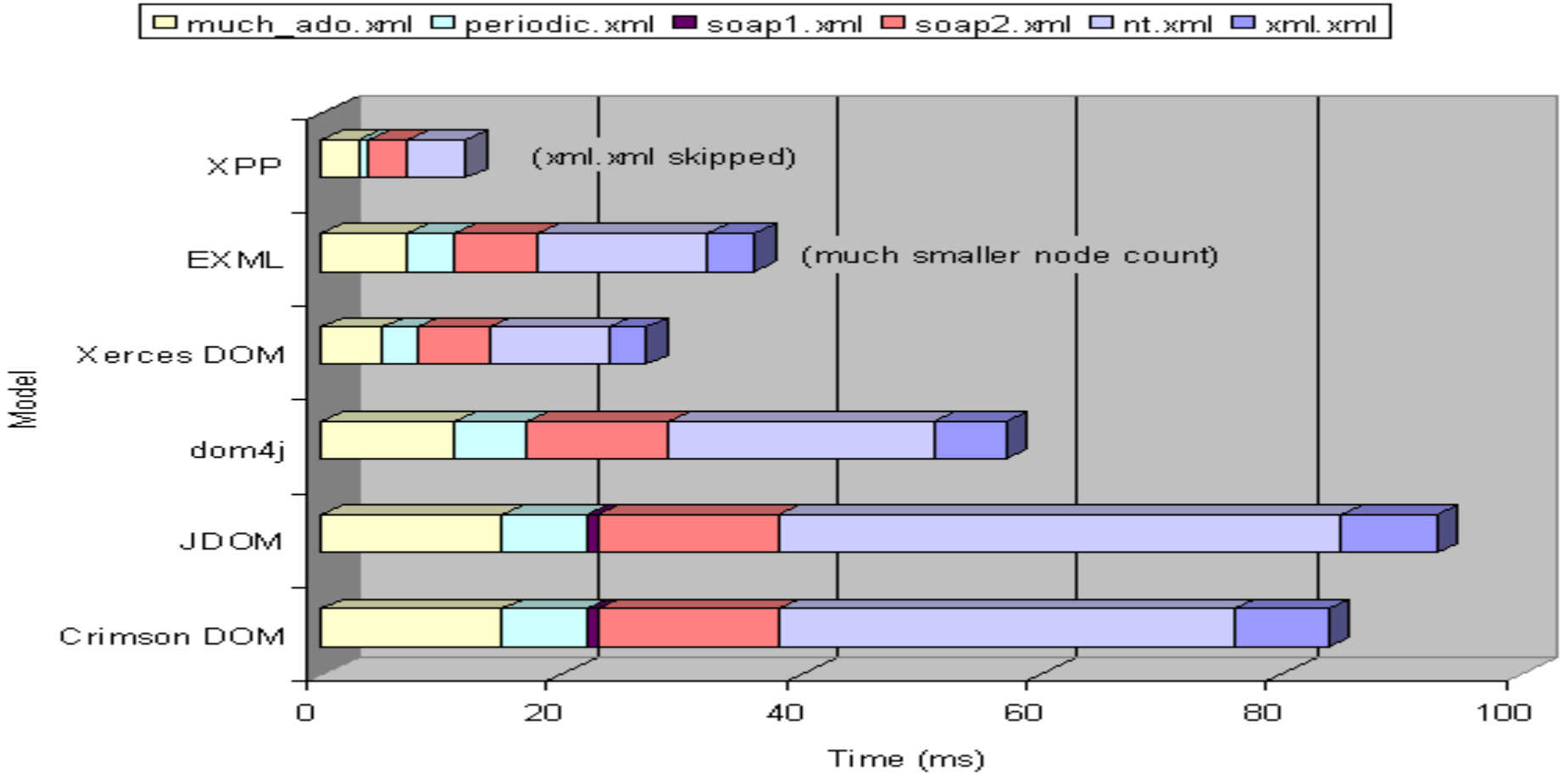
Document build time

- Czas potrzebny do sparsowania dokumentu tekstowego i skonstruowania reprezentacji dokumentu



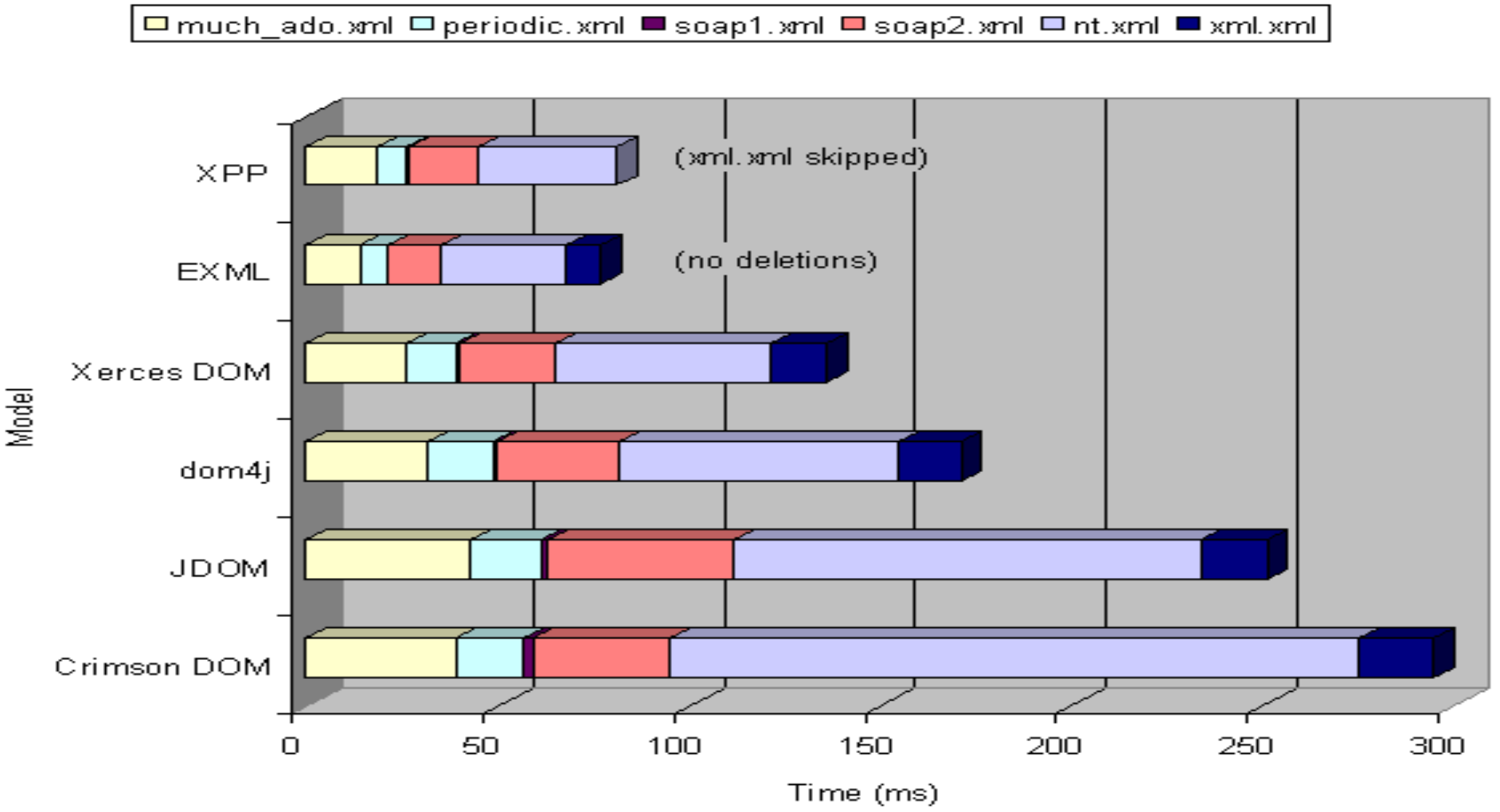
Document walk time

- Czas potrzebny do przejścia po już skonstruowanej reprezentacji dokumentu
- Test ważny dla aplikacji mających już sparsowany dokument, którym zależy na szybkim (częstym) dostępie do składowych
- W ogólności: czas przechodzenia jest szybszy od czasu parsowania



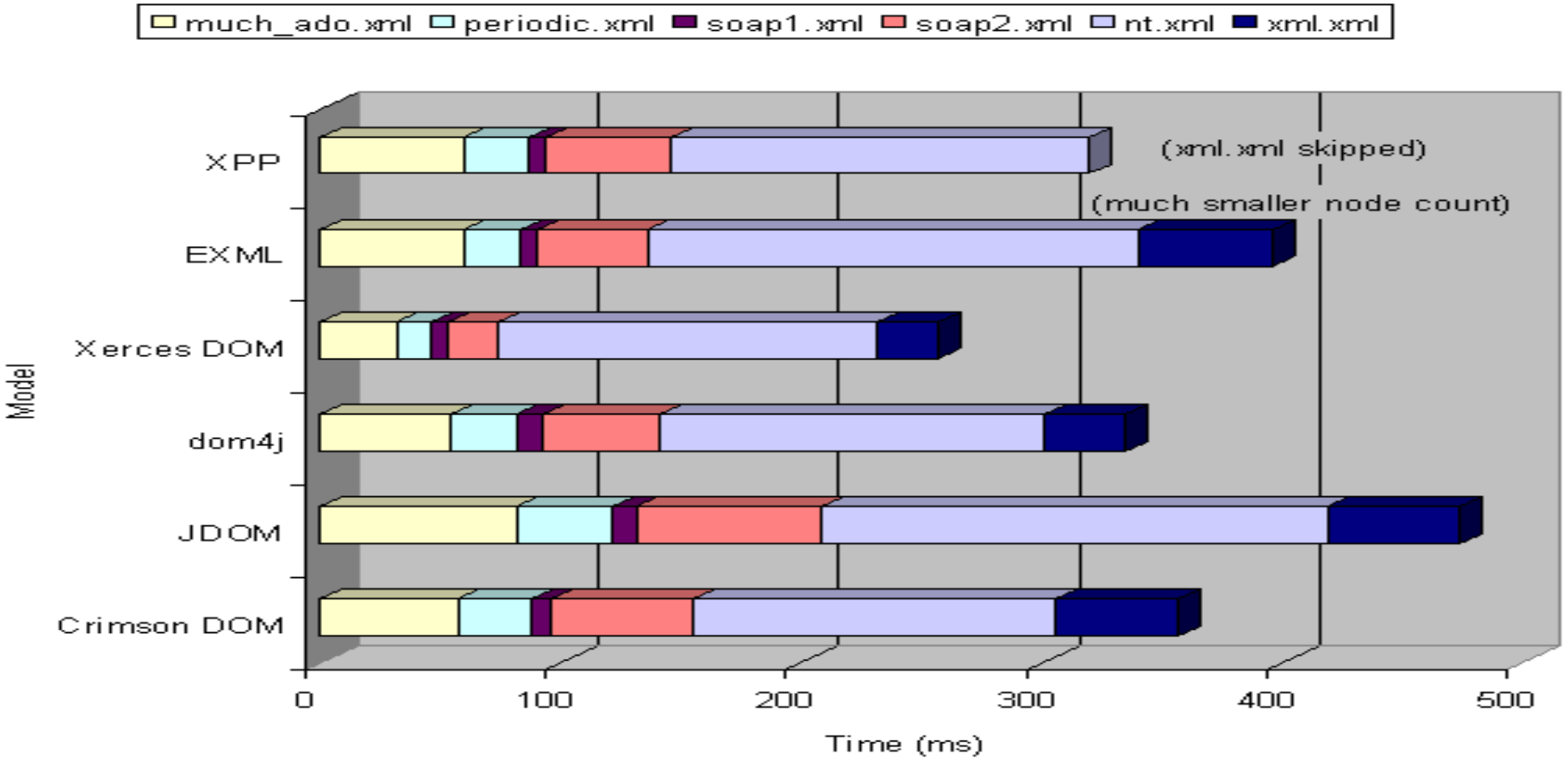
Document modify time

- W teście przechodzona jest cała reprezentacja dokumentu, usuwane są wszelkie fragmenty zawierające ciągi białych znaków, dla tekstu tworzone są całkiem nowe elementu zawierające ten tekst, dla każdego elementu dodawany jest atrybut



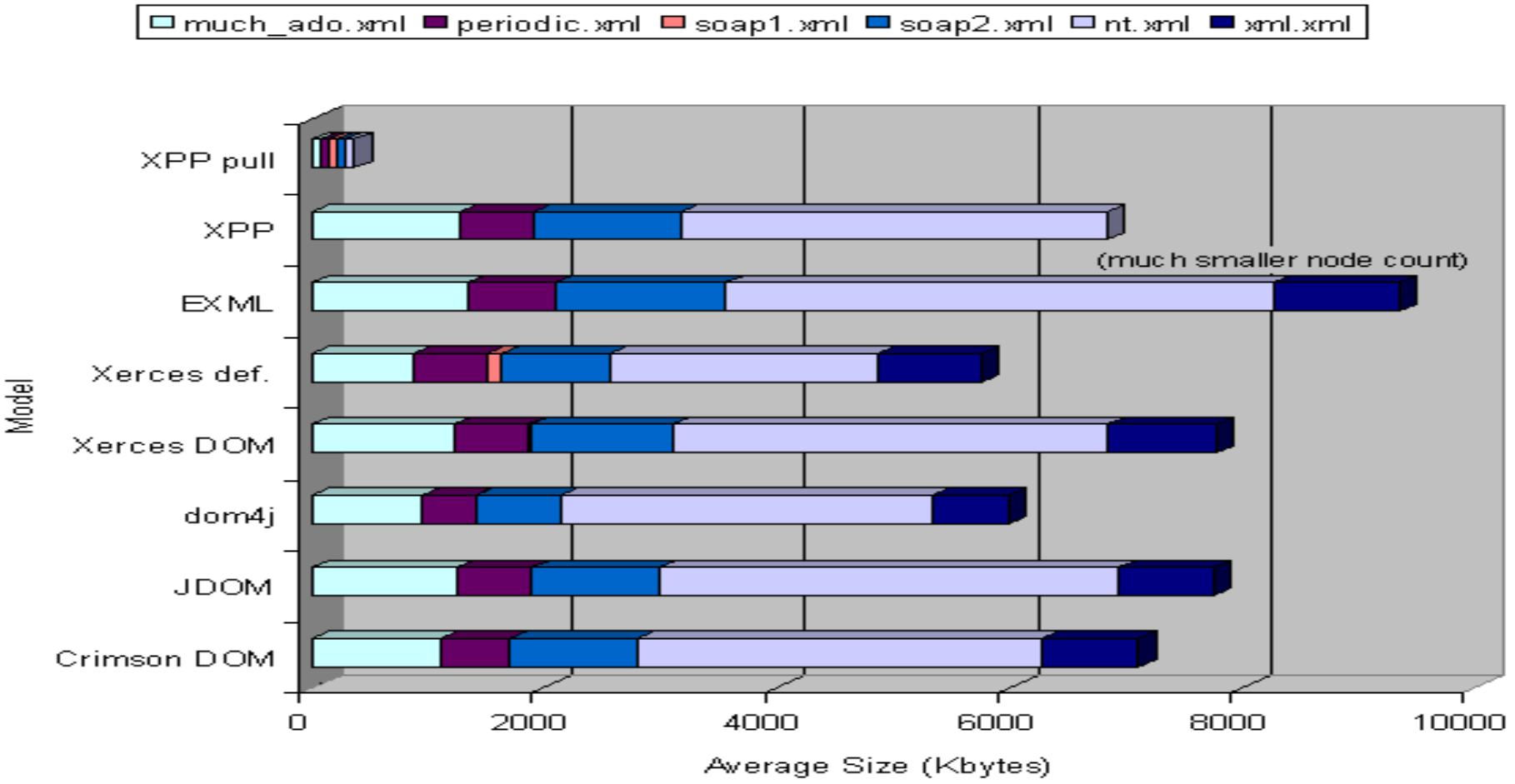
Text generation time

- Czas potrzebny na “odwrócenie” procesu parsowania (z reprezentacji dokumentu chcemy znowu otrzymać plik XML w postaci tekstowej)
- Test uruchamiany na standardowych, nie zmodyfikowanych, plikach
- Różne modele mogą dostarczać wiele sposobów zapisu dokumentu
- Widoczny jest znacznie mniejszy rozrzut wyników



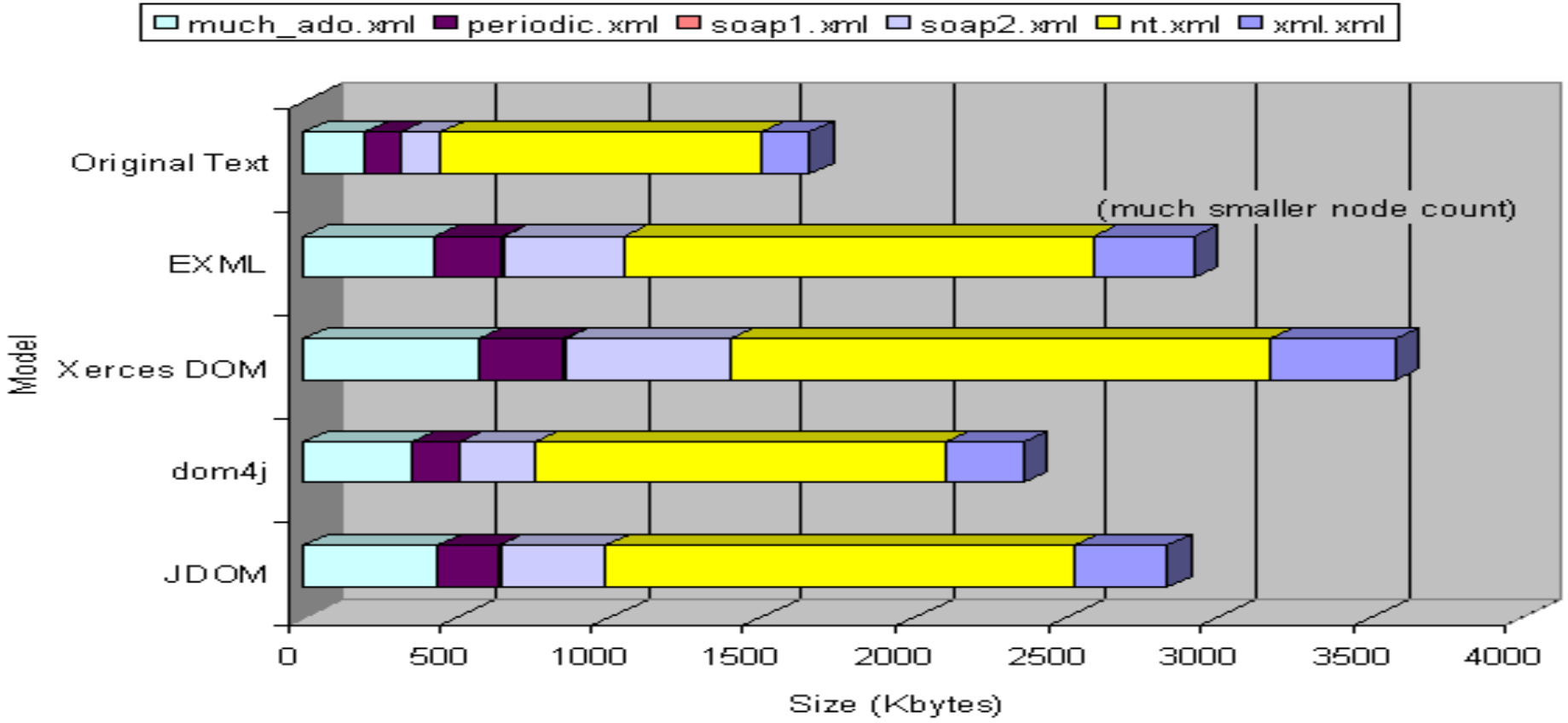
Document memory size

- Koszty pamięciowe przechowywania reprezentacji dokumentu
- Może to mieć znaczący wpływ przy pracy z dużymi dokumentami lub z wieloma dokumentami równocześnie
- Czasem koszty pamięciowe są ważniejszym kryterium niż poszczególne czasy
- soap1.xml jest tylko w jednej kopii
- Widoczna różnica między rozmiarem dokumentu a jego reprezentacji



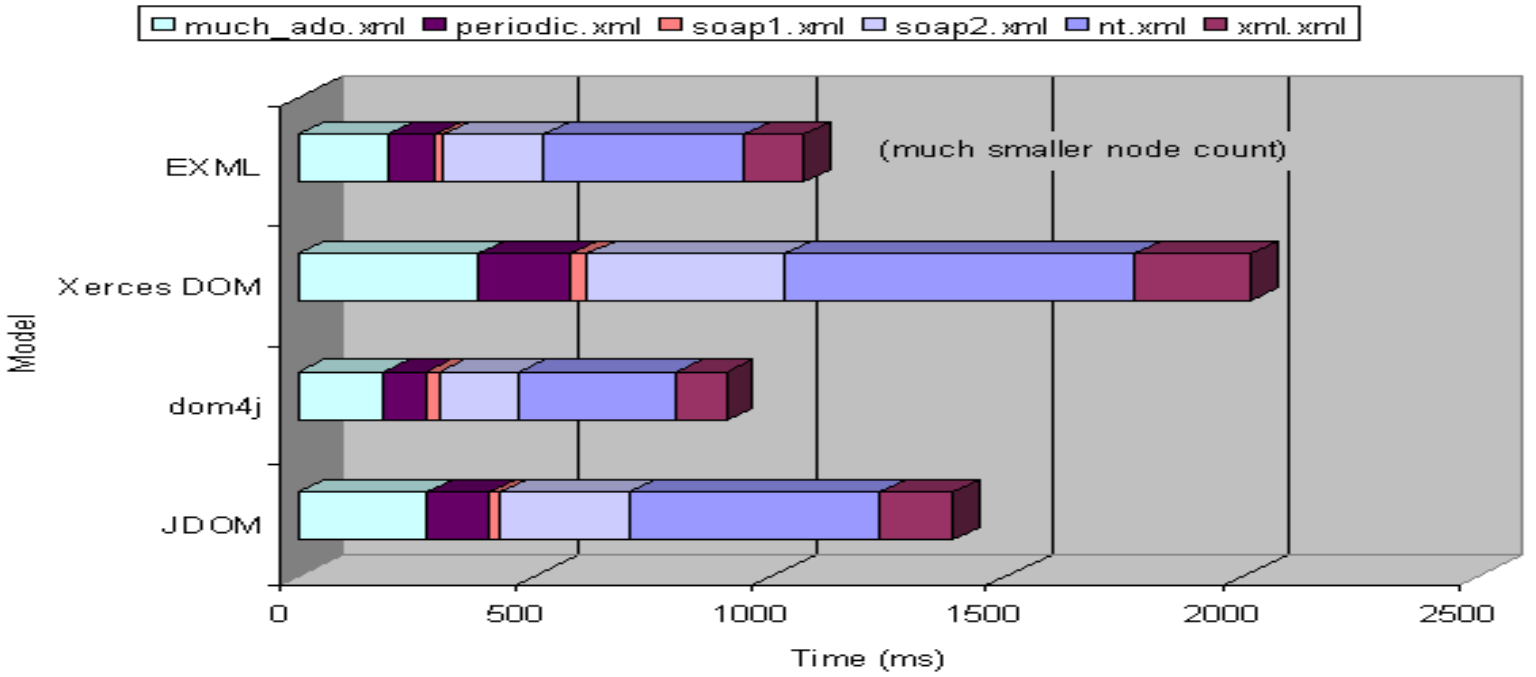
Java serialization (1/2)

- Test pokazuje czas potrzebny na serializacje obiektów w Javie dla po-szczególnych modeli
- Istotny dla aplikacji korzystających z np. Java RMI
- Lepiej dla wydajności (zarówno czasowej jak i pamięciowej) aby dokumenty były przesyłane w postaci tekstowej zamiast reprezentacji dokumentu

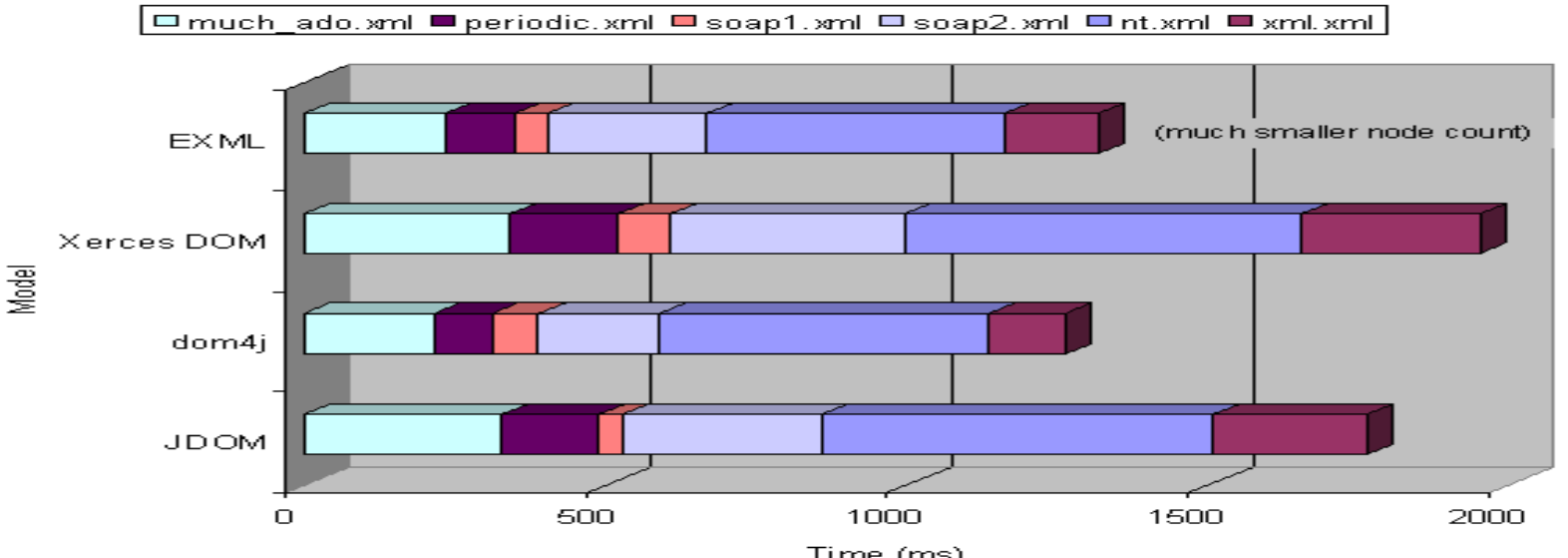


Java serialization (2/2)

- Output



- Input



Testy wydajnościowe - wnioski

- Różnice między testowanymi API w niektórych przypadkach są znaczące
- Zdecydowanym zwycięzcą jest XPP - idealny wybór jeśli nie przeszkadzają nam jego ograniczenia (brak walidacji, encji itd.)
- XPP może być szczególnie przydatny dla aplikacji uruchamianych jako aplety w przeglądarkach lub dla środowisk z ograniczoną pamięcią.
- Z modeli związanych z DOM najlepiej wypada dom4j
- Xerces DOM wypada całkiem nieźle ale zawodzi przy małych plikach i przy serializacji
- Najgorzej w teście wypadają JDOM (beta) oraz Crimson DOM

Podsumowanie

- Parser jest kawałkiem kodu, który czyta dokument i analizuje jego strukturę
- W chwili obecnej istnieje olbrzymi wybór parserów charakteryzujących się różnymi cechami
- SAX - oparty na zdarzeniach, w danej chwili "widzimy" tylko miejsce, w którym jesteśmy, nie możemy w żaden sposób cofnąć się do miejsca, w którym już byliśmy
- W3C DOM – model niezależny od języka programowania, przechowujący cały dokument w pamięci w postaci drzewa
- JDOM jest próbą stworzenia prostego i intuicyjnego modelu stworzonego tylko dla Javy i tylko dla XML
- W testach wydajnościowych najlepiej wypada parser XPP

Źródła

- http://www.oracle.com/technology/oramag/oracle/02-sep/o52jdom.html
- http://www.oracle.com/technology/oramag/oracle/02-nov/o62odev_jdom.html
- http://www.megginson.com/SAX
- http://www.w3.org/DOM

- "XML Kompendium programisty" Fabio Arciniegas
- "Processing XML with Java" Elliotte Rusty Harold
- "XML in a Nutshell, 2nd Edition" Elliotte Rusty Harold, W. Scott Means

Koniec.