



Web Engineering

Hauptstudium Informatik
Hauptstudium Medieninformatik
Bachelor/Master Informatik

WS 2004/2005
Stand: 17.12.2004

Michael Weber und Frank Kargl
Universität Ulm

Literatur



E. Harold, W. Means
XML in a Nutshell
Juli 2002 - 634 Seiten – O'Reilly
EUR 44,00
ISBN 0-596-00292-0



B. McLaughlin
Java & XML
April 2002 - 560 Seiten – O'Reilly
EUR 33,00
ISBN 3-89721-296-x

E.M. Burke
Java und XSLT
Mai 2002 - 520 Seiten – O'Reilly
EUR 33,00

ISBN 3-89721-296-x



Online-Ressourcen:

- <http://java.sun.com/xml/>
- <http://xml.apache.org/>
- <http://www.w3c.org/>
- <http://www.xml.com/> (O'Reilly Network)





6 XML Programmierung

6.1 Grundlagen

6.2 DOM

6.3 SAX

6.4 JAXB

Programmiermodelle für XML



- Textorientiert
 - Direktes Lesen/Manipulieren der Textdatei
 - Reguläre Ausdrücke, Suchen/Ersetzen etc.
 - Allgemeine Tools: z.B. vi, sed, awk, perl, ...
 - Standardwerte für Attribute? DTDs? Entities? Namensräume?
- Bewertung
 - Editieren und Lesen von Dokumenten mittels Texteditoren etc. häufig
 - Verarbeiten von XML Dokumenten mit allgemeinen Tools schwierig und fehleranfällig



Programmiermodelle für XML



- Ereignisorientiert
 - XML-Parser liest Dokument sequentiell
 - Beim Auftreten von bestimmten Bedingungen (z.B. Start Tag) werden Ereignisse generiert und via Callbacks an Applikation gesendet
 - Kontext (Namensräume, Gültigkeit usw.) wird mitgeführt
 - Immer nur ein Teil des Dokuments (+ Kontext) im Speicher
- Bewertung
 - Schnell und effizient
 - Kann auch sehr große Dokumente verarbeiten
 - Nie ganzes Dokument auf einmal verfügbar (z.B. zur Analyse von XPath Ausdrücken in Attributen)
 - Oft: Anwendung selbst speichert Zustand



Programmiermodelle für XML



- Baum- oder objektorientiert
 - XML-Parser liest komplett ein
 - Interne Darstellung als Baum- oder Objektstruktur
 - Zugriff durch Navigationsmethoden
- Bewertung
 - Verbraucht evtl. viel Speicher
 - Erlaubt freie Navigation im Baum
 - Unterstützung von XPath u.U. bereits integriert



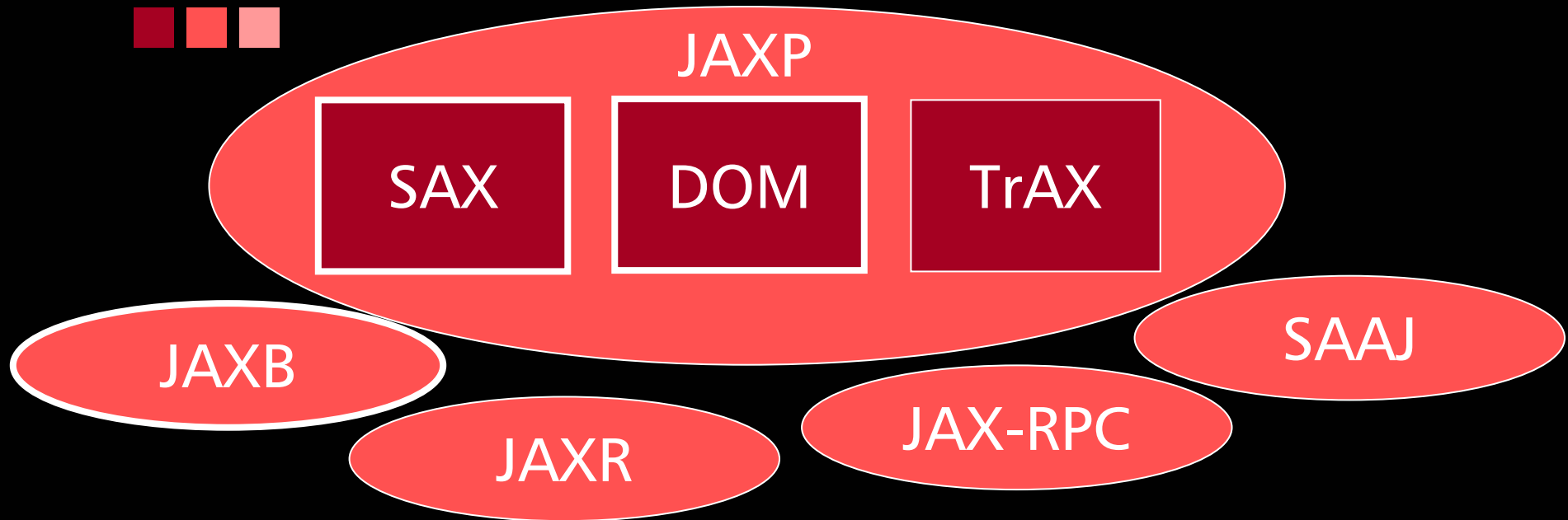
Programmiermodelle für XML



- Transformation
 - Z.B. mittels XSLT
 - Keine Programmierung im engeren Sinne
- Implizit
 - Z.B. bei XML-RPC
 - Kein direkter Kontakt des Programmierers mit XML
- Frameworks
 - z.B. Cocoon, Struts
 - Für dedizierte Aufgabenbereiche
- Bewertung
 - Für Spezialaufgaben durchaus geeignet
 - Keine Einarbeitung in XML notwendig(?)



Übersicht APIs



- JAXP: Java API for XML Processing
 - SAX: Simple API for XML
 - DOM: Document Object Model
 - TrAX: Transformation API for XML
- JAXB: Java API for XML Binding
- JAXR: Java API for XML Registries
- JAX-RPC: Java API for XML Remote Proc. Call
- SAAJ: SOAP with Attachments API for Java





6 XML Programmierung

6.1 Grundlagen

6.2 DOM

6.3 SAX

6.4 JAXB

Document Object Model



- Programmiersprachen-unabhängiges Modell des W3C zum Zugriff auf Markup-Dokumente
- Allgemeiner und (HTML- bzw. XML-) spezifischer Teil
- Beschreibung der Schnittstellen in Corba-IDL
- Language-Bindung für Java, C, C++, Scheme uvm.
- DOM XML Baum ähnlich wie bei XPath



DOM Grundlagen



- Initialisieren und Parsen eines Dokuments

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
DocumentBuilder builder = factory.newDocumentBuilder();  
document = builder.parse(new File(args[0]));
```

- Parametrisieren

```
factory.setValidating(true);  
factory.setIgnoringElementContentWhitespace(true);  
factory.setNamespaceAware(true);  
factory.setIgnoringComments(true);
```



DOM Grundlagen



■ Zugriff auf Knoten

```
static void printTree(Node node, int level) {
    for (int i = 0; i < level; i++) {System.out.print('-');}
    System.out.print(node.getNodeName());
    if (node.getNodeValue() != null) {
        System.out.print(": " + node.getNodeValue());
    }
    if (node.hasAttributes()) {
        System.out.print(" # ");
        NamedNodeMap attributes = node.getAttributes();
        for (int j = 0; j < attributes.getLength(); j++) {
            Node attribute = attributes.item(j);
            System.out.print(attribute.getNodeName()+"="+
                attribute.getNodeValue()+";");
        }
    }
    System.out.println();
    NodeList nl = node.getChildNodes();
    for (int k = 0; k < nl.getLength(); k++) {
        Node subNode = nl.item(k);
        printTree(subNode, level + 1);
    }
}
```



DOM Entwicklung



- DOM Level 1
 - Grundsätzliche Verarbeitung von HTML/XML Dokumenten
- DOM Level 2
 - Modularisierung
 - Views, Events, Style, Traversal, Range
- DOM Level 3
 - Ein-/Ausgabe von Dokumenten
 - Integration von Validierung
 - DTDs und XML Schema
 - XPath





6 XML Programmierung

6.1 Grundlagen

6.2 DOM

6.3 *SAX*

6.4 JAXB

Simple API for XML



- Industriestandard der XMLDEV Mailingliste für XML Parsing in Java
- Ereignisorientiert
- Mittlerweile auch Umsetzungen für andere Programmiersprachen (C++, Python, Perl, Eiffel, ...)
- Wichtig für DOM Programmierer:
DOM Fehlerbehandlung benutzt SAX Exceptions



SAX Grundlagen



■ Einlesen eines Dokuments

```
SAXParserFactory spf = SAXParserFactory.newInstance();  
SAXParser sp = spf.newSAXParser();  
XMLReader reader = sp.getXMLReader();
```

```
InputStream inputStream = new InputStream(uri);  
reader.parse(inputStream);
```

■ Setzen der Callback-Handler

```
public class SimpleSAX implements ContentHandler,  
    ErrorHandler, LexicalHandler {  
    ...  
    reader.setContentHandler(this);  
    reader.setErrorHandler(this);  
    ...  
}
```



SAX Features und Properties



- Steuern das Verhalten des Parsers
 - Features: Binär
 - Properties: Wert

```
String featureURI = "http://xml.org/sax/features/validation";
reader.setFeature(featureURI, true);
boolean state = true;
String featureURI = "http://xml.org/sax/features/namespace-prefixes";
reader.setFeature(featureURI, state);
featureURI = "http://xml.org/sax/features/namespace-prefixes";
reader.setFeature(featureURI, !state);
reader.setProperty("http://xml.org/sax/properties/lexical-handler",
    this);
```



SAX Callback Handler



- ContentHandler
- DefaultHandler liefert leere Methoden (für alle Callback Handler)

```
public void setDocumentLocator(Locator locator) {}
public void startDocument() throws SAXException {}
public void endDocument() throws SAXException {}
public void startPrefixMapping(String prefix, String uri) {}
public void endPrefixMapping(String prefix) {}
public void startElement(String namespaceURI, String localName,
    String qName, Attributes atts) {}
public void endElement(String namespaceURI, String localName,
    String qName) {}
public void characters(char ch[], int start, int length) {}
public void ignorableWhitespace(char ch[], int start, int length) {}
public void processingInstruction(String target, String data) {}
public void skippedEntity(String name) {}
```



SAX Callback Handler



- **ErrorHandler**

```
public void warning(SAXParseException exception) {}  
public void error(SAXParseException exception) {}  
public void fatalError(SAXParseException exception){}
```

- **LexicalHandler**

```
public void startDTD(String name, String publicId, String systemId) {}  
public void endDTD() {}  
public void startEntity(String name) {}  
public void endEntity(String name) {}  
public void startCDATA() {}  
public void endCDATA() {}  
public void comment(char ch[], int start, int length) {}
```

- **Weitere Features: Filter, Transformer ...**





6 XML Programmierung

6.1 Grundlagen

6.2 DOM

6.3 SAX

6.4 JAXB

JAXB Einführung



- Java API for XML Binding
- SAX und DOM
 - generische Parser für beliebiges (wohlgeformtes HTML)
- JAXB
 - generiert spezifischen Parser für bestimmte Dokumentenklasse
 - auf Basis von XML-Schema (Auszug)

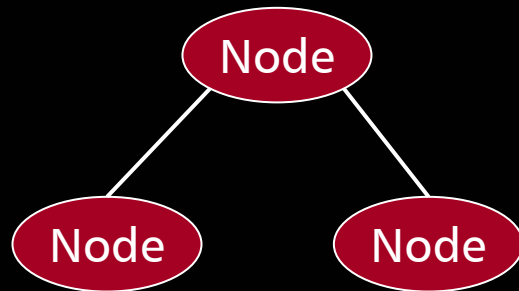


DOM vs. JAXB

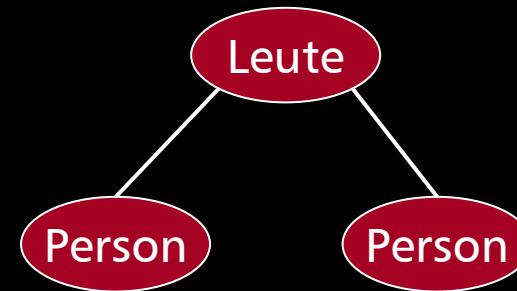


```
<?xml version="1.0"?>
<leute>
  <person> ... </person>
  <person> ... </person>
</person>
</leute>
```

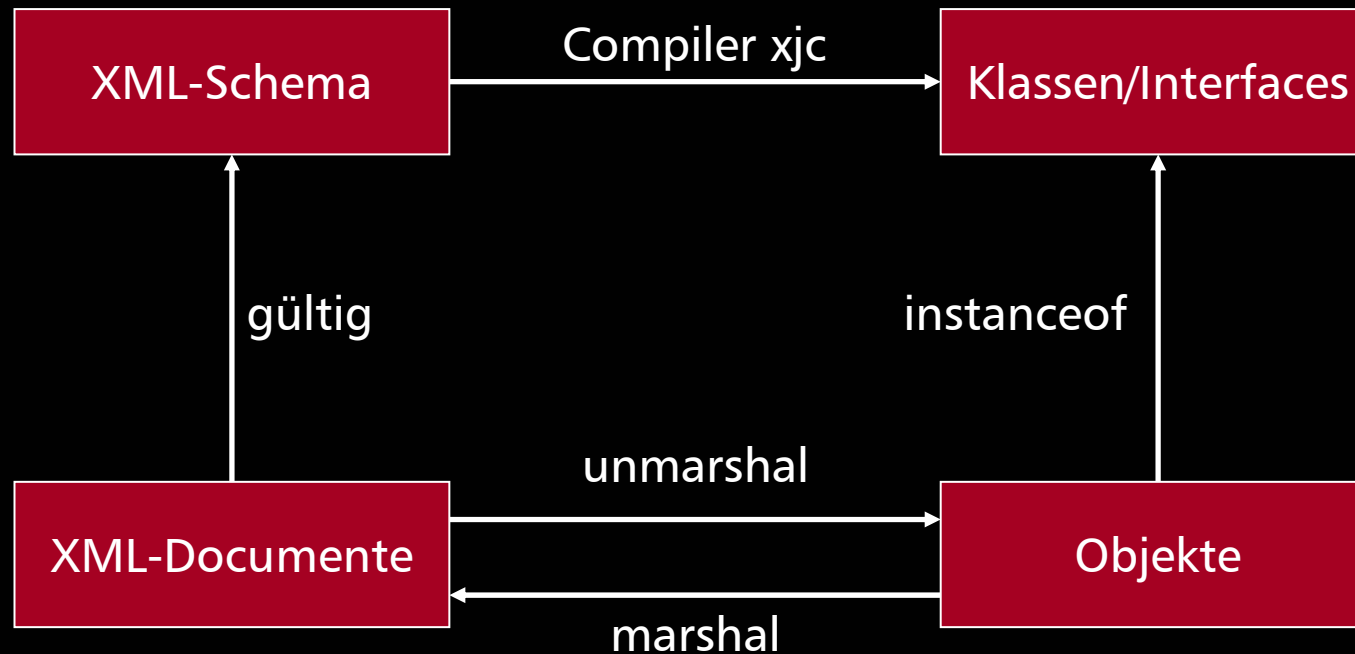
DOM



JAXB



JAXB Prinzip



JAXB Bindings



- Default Bindings

- XML Namen → Java names
- Atomare Schema Typen → vordefinierte Javatypes
- z.B.

`xs:string` → `java.lang.string`

`xs:decimal` → `java.math.BigDecimal`

`xs:int` → `int`

`xs:QName` → `javax.xml.namespace.QName`

`xs:date` → `java.util.Calendar`

- Strukt. Schema Typen → generierte Java Interfaces
- `javax.xml.bind` – Framework Interfaces
- Binding Customization



JAXB Beispiel



schema.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" jaxb:version="1.0">
  <xs:element name="simple" type="xs:int" />
</xs:schema>
```

simple.java

```
package schema;
public interface Simple extends javax.xml.bind.Element {
    int getValue();
    void setValue(int value);
}
```

xjc schema.xsd -p schema

simpleImpl.java

```
package schema.impl;
public class SimpleImpl implements schema.Simple,... {
    protected int _Value;
    public SimpleImpl(int value) {...}
    public int getValue() { return _Value; }
    public void setValue(int value) { Value = value; has_Value = true; }
    ...
}
```

+ viele weitere Klassen



JAXB in Aktion



useSimple.java

```
...
JAXBContext jc = JAXBContext.newInstance("simple",
    this.getClass().getClassLoader());
Marshaller mar = jc.createMarshaller();
mar.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, (Object) new
    Boolean(true));
mar.setProperty(Marshaller.JAXB_NO_NAMESPACE_SCHEMA_LOCATION,
    (Object) new String("schema.xsd"));
OutputStream os = new FileOutputStream("ausgabe.xml");
ObjectFactory of = new simple.ObjectFactory();
Simple simple = (Simple) of.createSimple();
simple.setValue(5);

Validator val = jc.createValidator();
val.validate(simple);
mar.marshal(simple, os);
os.close();
...
```

ausgabe.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<simple xsi:noNamespaceSchemaLocation="schema.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">5</simple>
```



Beispiel: Sequenz und Wiederholung



```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title"
        type="xs:string"/>
      <xs:element name="author"
        type="xs:string"
        maxOccurs="unbounded"/>
      <xs:element name="pages"
        type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
interface Book extends BookType,
  javax.xml.bind.Element {}

interface BookType {
  String getTitle();
  void setTitle(String x);
  List getAuthor();
  int getPages();
  void setPages(int x);
}
```



Beispiel: choice



```
<xs:complexType
  name="volume">
  <xs:choice>
    <xs:element
      ref="book"/>
    <xs:element
      ref="journal"/>
  </xs:choice>
</xs:complexType>
```

```
interface Volume {
  BookType getBook();
  void setBook(BookType);

  JournalType getJournal();
  void setJournal(JournalType);
}
```



Customization: isSet () Methoden



```
<xs:annotation><xs:appinfo>  
  <jxb:globalBindings generateIsSetMethod="true"/>  
</xs:appinfo></xs:annotation>
```

```
<xs:complexType  
  name="volume">  
  <xs:choice>  
    <xs:element  
      ref="book"/>  
    <xs:element  
      ref="journal"/>  
  </xs:choice>  
</xs:complexType>
```

```
interface Volume {  
  BookType getBook ();  
  void setBook (BookType);  
  boolean isSetBook ();  
  
  JournalType getJournal ();  
  void setJournal (JournalType);  
  boolean isSetJournal ();  
}
```



Vorteile und Nachteile



- Vorteile:
 - JAXB *benötigt* ein *XML-Schema*
 - stellt die Gültigkeit der XML Dokumente sicher
 - JAXB Parser ist *schneller* als ein generischer SAX Parser
 - Ein mit JAXB generierter Baum ist *kleiner* als ein DOM Baum
 - Ein JAXB Baum lässt sich *leichter* in Applikationen verwenden
 - Man kann den Baum *verändern* und wieder als XML *speichern*
- Nachteile:
 - JAXB *benötigt* ein *XML-Schema*
 - man kann JAXB nicht verwenden, um *generisches* XML zu verarbeiten (z.B. für einen XML Editor)
 - Umfangreichere Vorarbeiten, um passenden JAXB Parser zu generieren
 - zahlt sich möglicherweise aus, da Applikation viel einfacher wird

