Programming Mobile Devices

XML Parsing

University of Innsbruck

WS 2009/2010

thomas.strang@sti2.at

STI · INNSBRUCK

You all have seen typical XML files, e.g.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<bibliography>
  <phdthesis author="Thomas Strang">
    <title>
      Service-Interoperability in
      Ubiquitous Computing Environments
    </title>
    <isbn>3-8007-2823-0</isbn>
  </phdthesis>
  <!-- … several more … -->
</bibliography>
```

biblio.xml

Well-Formed vs. Valid

- An XML document is **well-formed**, if it fulfills the following criteria:
  - There is exactly one root element, containing any other element
  - For all elements exist a opening and a closing tag
  - Nesting is not broken anywhere
  - Any attribute has a value enclosed by „“ or ‚‘
  - A characterset has to be defined in the header if not default UTF

- An XML document is **valid**, if it has an associated document type declaration and if the document complies with the constraints expressed in it. [http://www.w3.org/TR/REC-xml/#dt-valid]

XML Schema

- XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide mechanisms for defining the structure, content and [to some extend] semantics of XML documents
- Idea: Specify Syntax & Grammar of instance documents using XML itself, e.g. similar to

```
<schema>
  <element name="name" type="string" />
  <element name="qualification" type="string" />
  <element name="born" type="date" />
  <element name="dead" type="date" />
  <element name="isbn" type="string" />
  <element name="id" type="ID" />
  <element name="available" type="boolean" />
  <element name="lang" type="language" />
</schema>
```

Instance vs. Schema

- Instance

```
<title lang="en">
  Being a dog is a full-time job
</title>
```

- Simple Content
- Complex Type (Attribute!)

- Schema

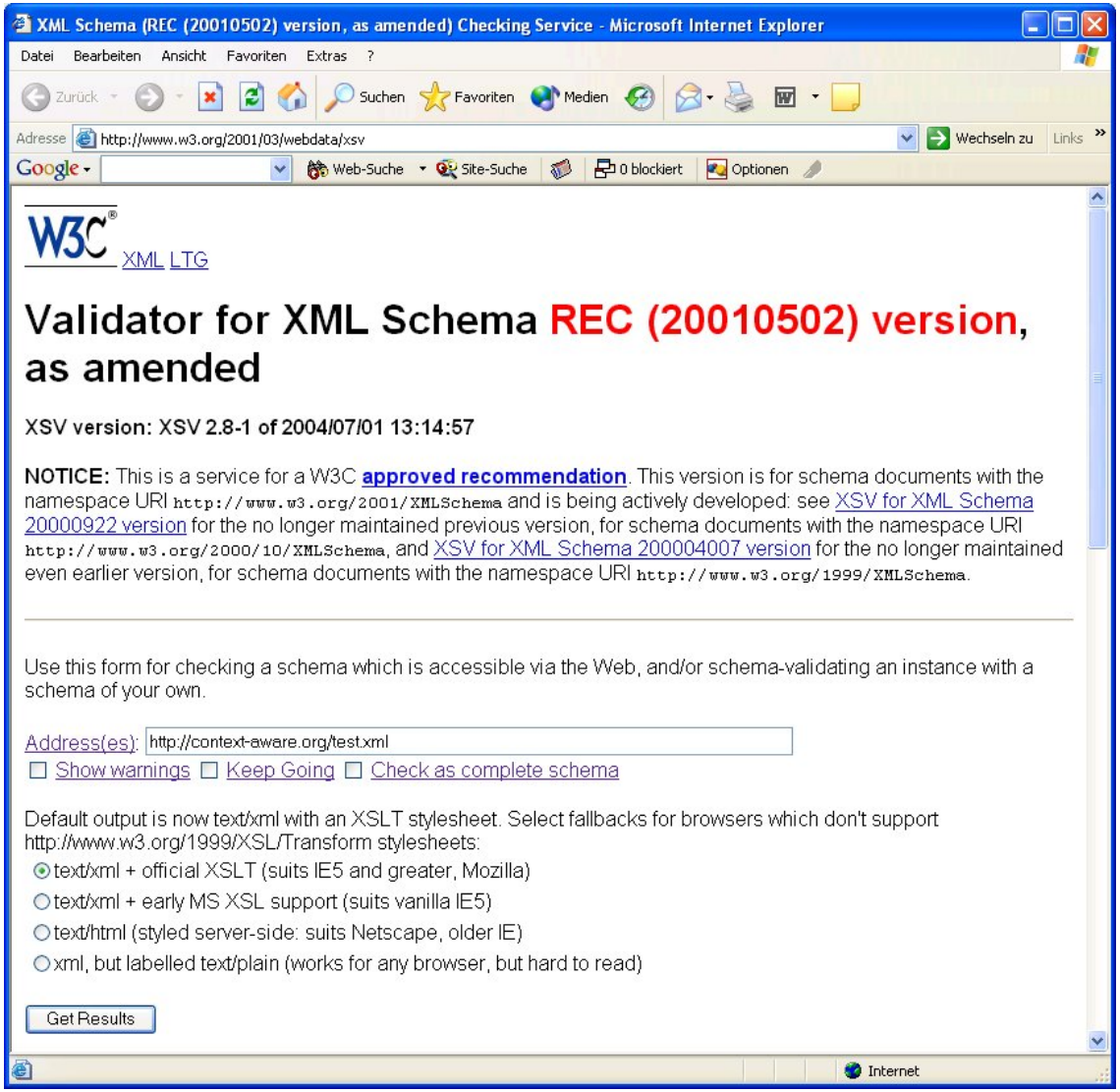
```
<xs:element name="title">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="lang" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

"the element named `title` has a complex type which is a simple content obtained by extending the predefined datatype `xs:string` by adding the attribute defined in this schema and having the name `lang`"

Validation

- Validation of
  - Schema itself against XML Schema Spec
  - Instance against Schema
- Validators
  - http://www.w3.org/2001/03/webdata/xsv
  - http://www.stg.brown.edu/service/xmlvalid

Validation with XSV

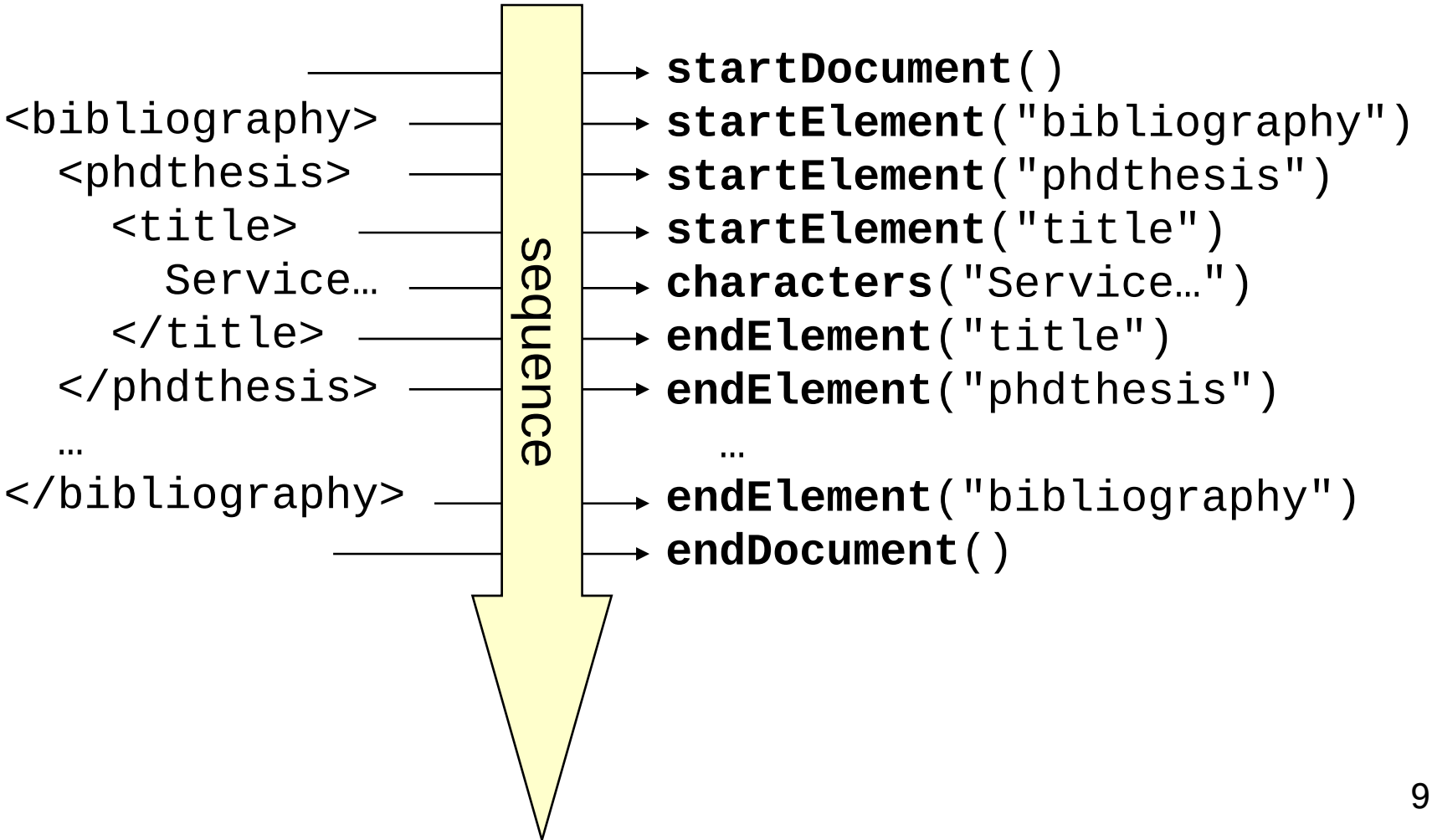


Parser

- Parsers are used to read XML documents into a programming language specific data structure
- Parsers may be
  - validating or non-validating
  - processing model: event-based or document-based
- Big differences between the processing models w.r.t.
  - memory requirements
  - access to the elements

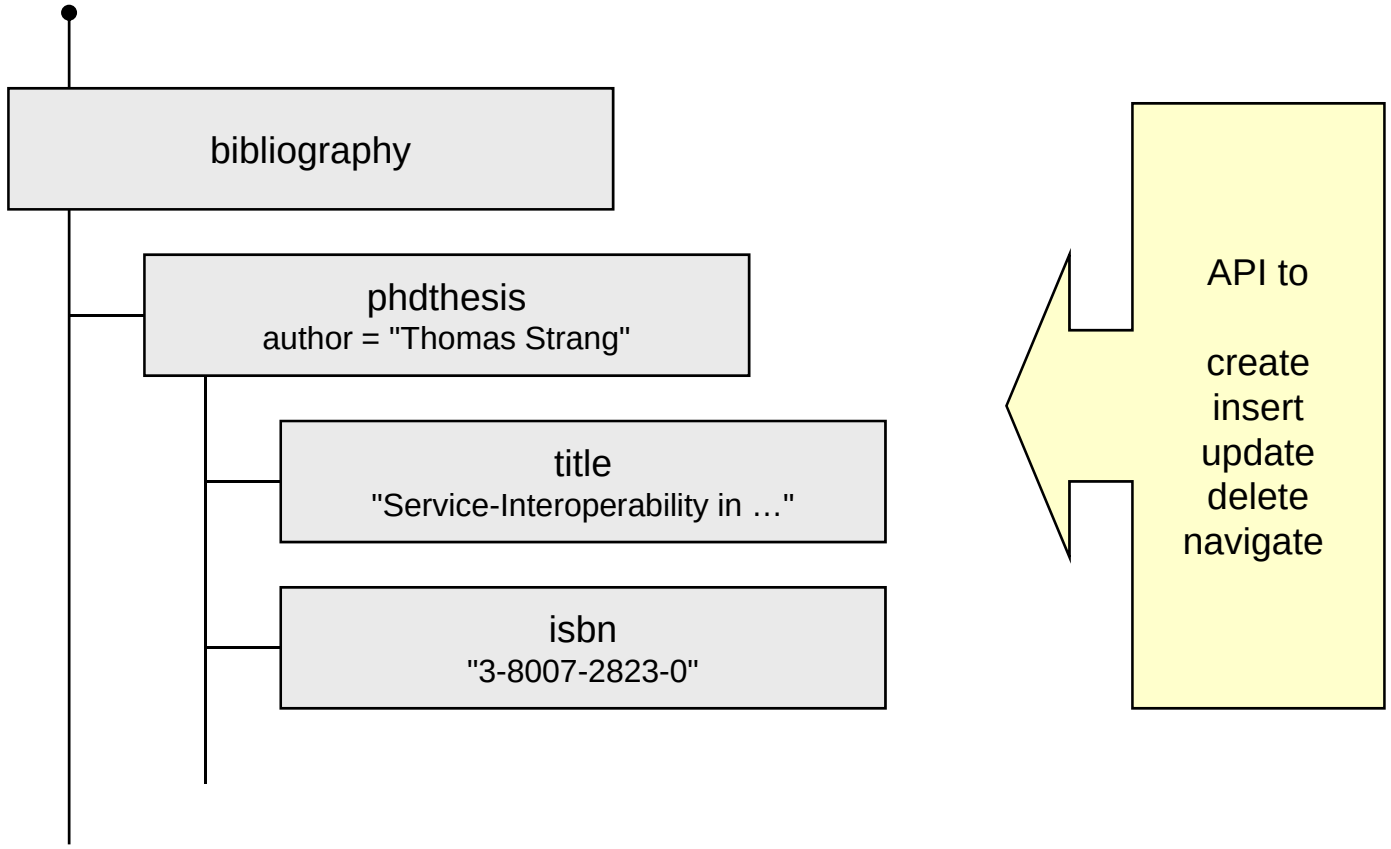
SAX Parser

- Serial Access to XML (SAX) – event-based



DOM Parser

- Document Object Model (DOM) – document-based
  - defined by interfaces (no classes)

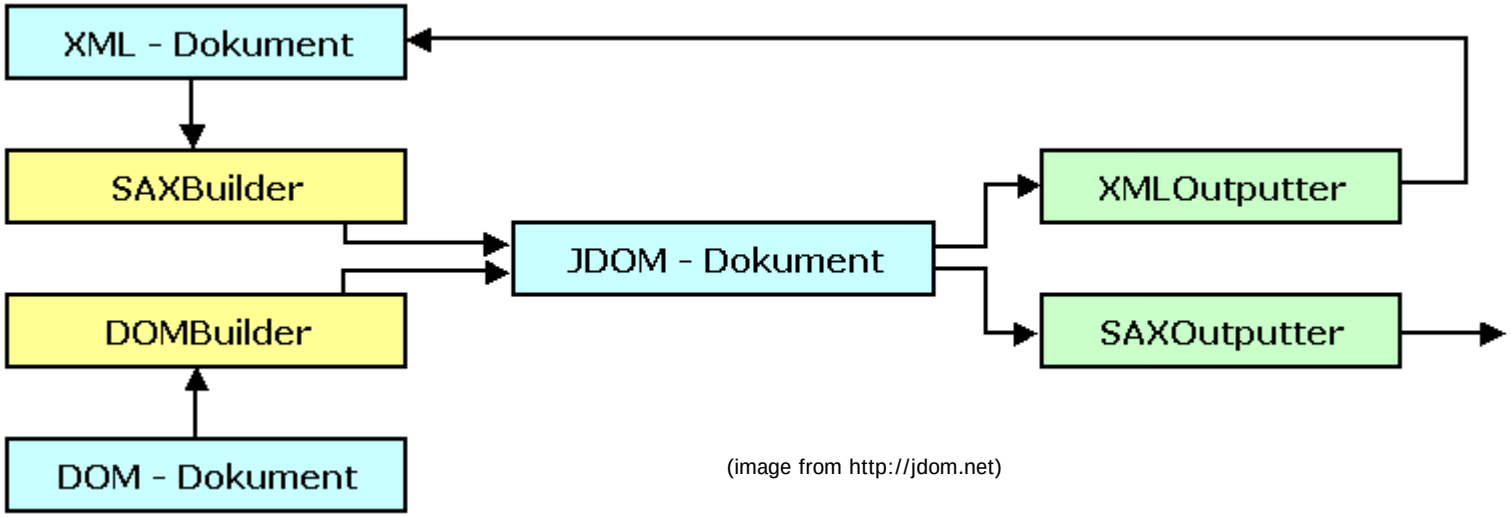


Using DOM (J2SE)

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;

Document doc = null;
try {
  DOMParser p = new DOMParser();   // instantiate Parser

  // activate validation
  p.setFeature("http://xml.org/sax/features/validation", true);

  p.parse("http://www.deri.at/teaching.xml");  // parse file
  doc = p.getDocument();           // get document from parser
}
catch (IOException io) {           // z. B. file error
  ...
}
catch (SAXException s) {           // z. B. invalid XML
  ...
}
```

J2SE Example: JDOM

- open source Java API for DOM
- "Natural" way for Java programmers to access XML
  - uses collection classes, overloading, reflection
  - defined as classes *and* interfaces
- Cooperative, not competitive to SAX and DOM:

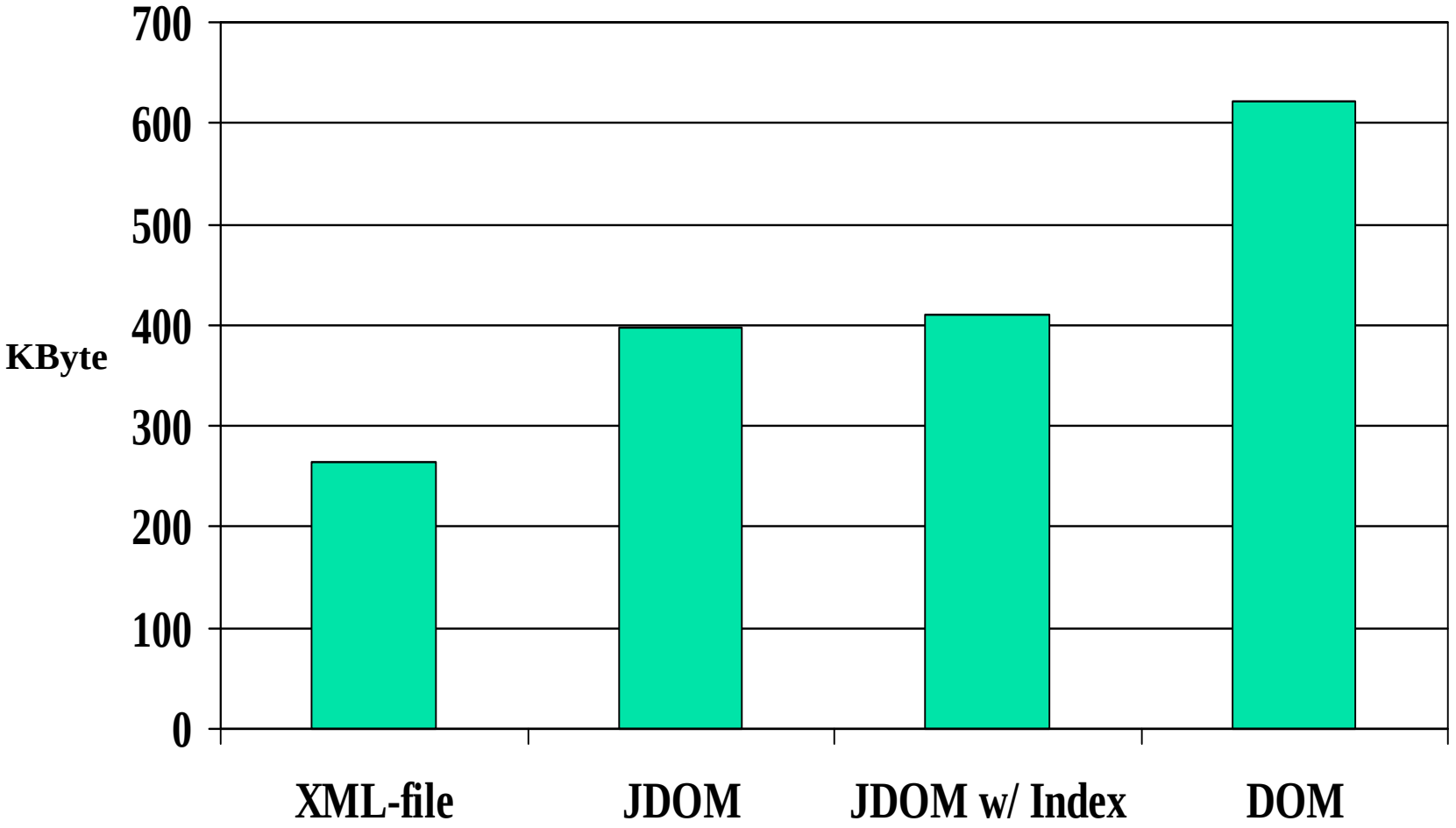


(image from http://jdom.net)

Using JDOM (J2SE)

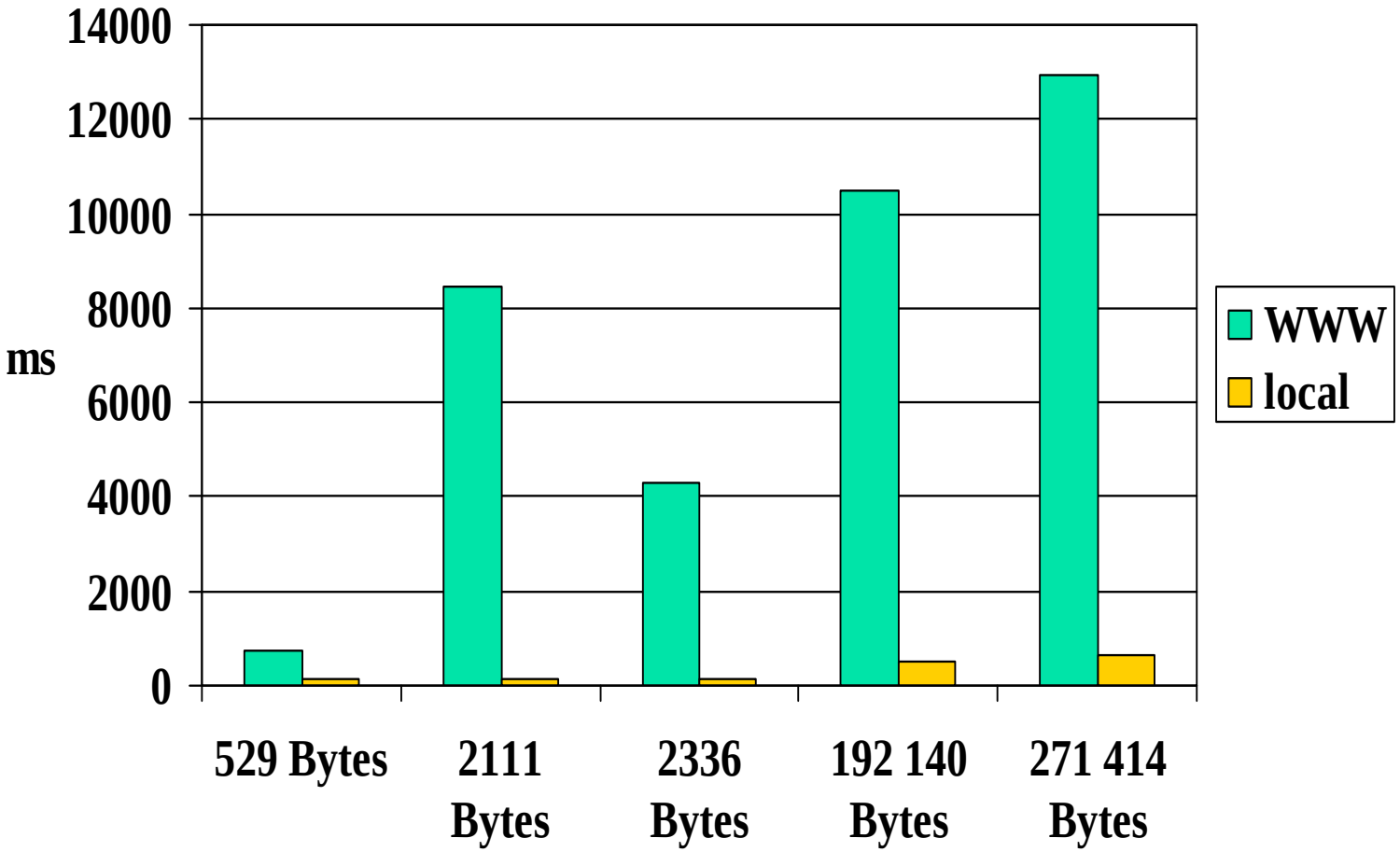
```
import org.jdom.input.SAXBuilder;
import org.jdom.Document;

Document doc = null;
try {
  SAXBuilder b = new SAXBuilder(true /*validating*/ );
  doc = b.build("http://www.deri.at/teaching.xml");
  // ...
  doc.addContent(new Element("pmd-lecture")
        .addAttribute("MSc","true")
        .addContent(new Element("unit")
          .setText("XML parsing")))
}
catch (JDOMException j) {
}
```

Memory Requirements



[Bielert, 2001]

Access Speed: Improvements with dynamic content



(given a set of diverse XML files from different WWW servers)

[Bielert, 2001]

J2ME: Smaller, smaller, smaller - kXML

- kXML is a small XML event based parser, specially designed for constrained environments such as Applets, Personal Java or MIDP devices
- kXML 2 implements *pull based XML parsing* (see http://xmlpull.org), which combines some of the advantages of SAX (push) and DOM
- source is provided on sourceforge at http://kxml.sourceforge.net

Pull Parsing

- In contrast to push parsing, pull parsing lets the programmer "pull" the next event from the parser.
- In push parsing you would have to maintain the state of the current part of the data you were parsing, and based on the events passed to the listener you would have to take care to restore any previous state variables and save new ones when you were changing to a different state.
- Pull parsing makes it easier to deal with state changes because you can pass parser to different functions, which can maintain their own state variables.

kXML 1: Push Parsing (like SAX)

```
private void kXML1traverse(XmlParser parser) {   // avoid recursion!!
    boolean leave = false;
    do {
        ParseEvent event = parser.read();
        switch (event.getType())
        {
        // reads out the start-tags and attributes of the xml-stream
        case Xml.START_TAG:
            //... prepare data structures dependent on type of element
            break;
        // reads out the text-data
        case Xml.TEXT:
            //... fill current data structure
            break;
        // reads out the end-tag
        case Xml.END_TAG:
            //... evaluate current data structure
            break;
        // reads out the end-document-tag
        case Xml.END_DOCUMENT:
            leave = true;
            break;
        default:        // do nothing
        }
    } while (!leave);
```

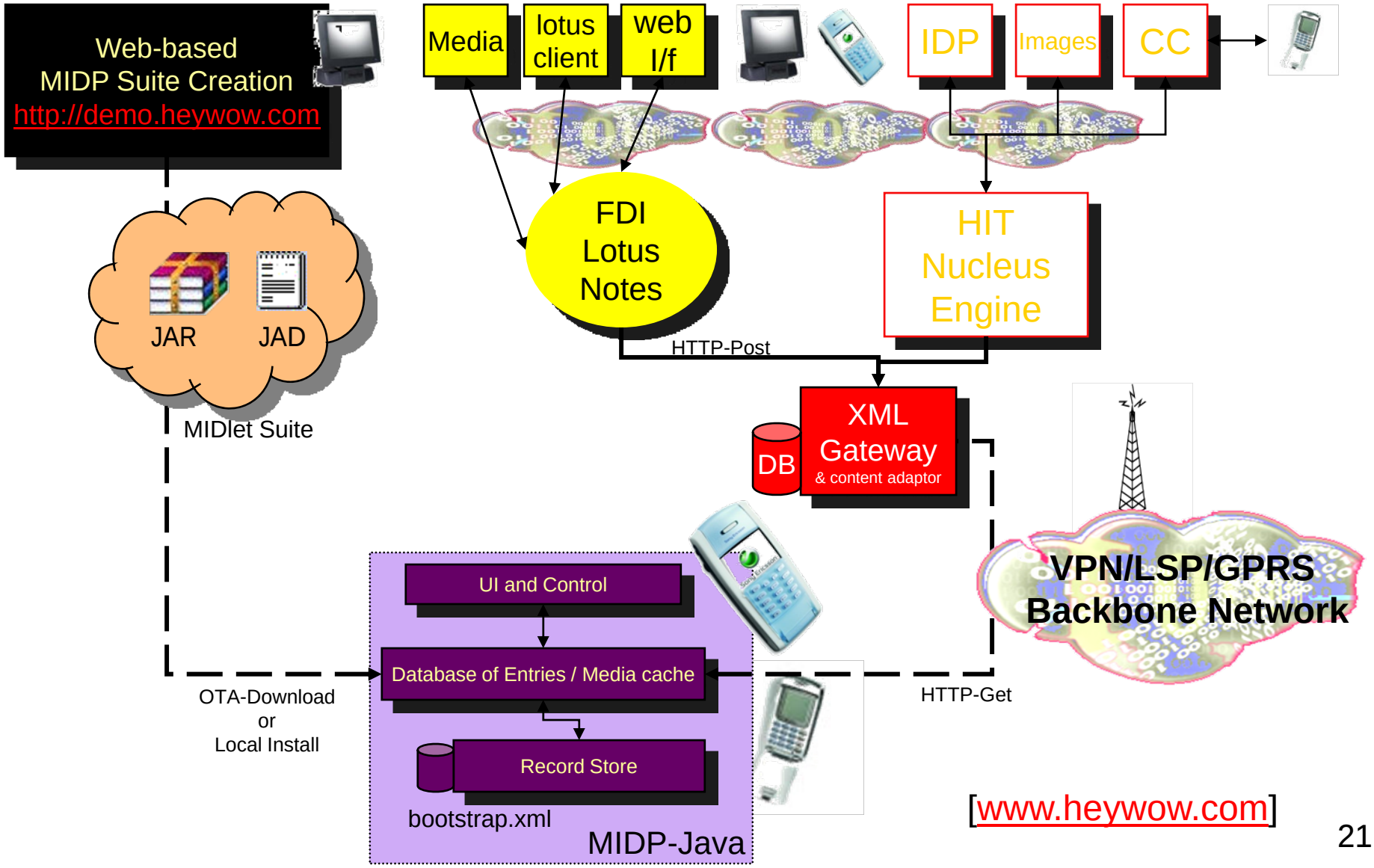
kXML 2: Pull Parsing

```
XmlParser parser =
   new XmlParser(
      new InputStreamReader(
         this.getClass().getResourceAsStream("file.xml")));
//...

while ((event = parser.read()).getType() != Xml.END_DOCUMENT) {
   if (name != null && name.equals("address"))
      parseAddressTag( parser );
//...

parseAddressTag(XmlParser parser) {
while ((event = parser.peek()).getType() != Xml.END_DOCUMENT) {
   if (type == Xml.END_TAG && name.equals("address")) return;
   ParseEvent next = parser.read();
   if (next.getType() != Xml.TEXT) continue;
   System.err.println(name + ": " + text);
//...
```

JSR-172

- Alternative to kXML if supported by the Phone: JSR-172
  - XML parser following the SAX-Model
  - as part of Web Service API (WSA)

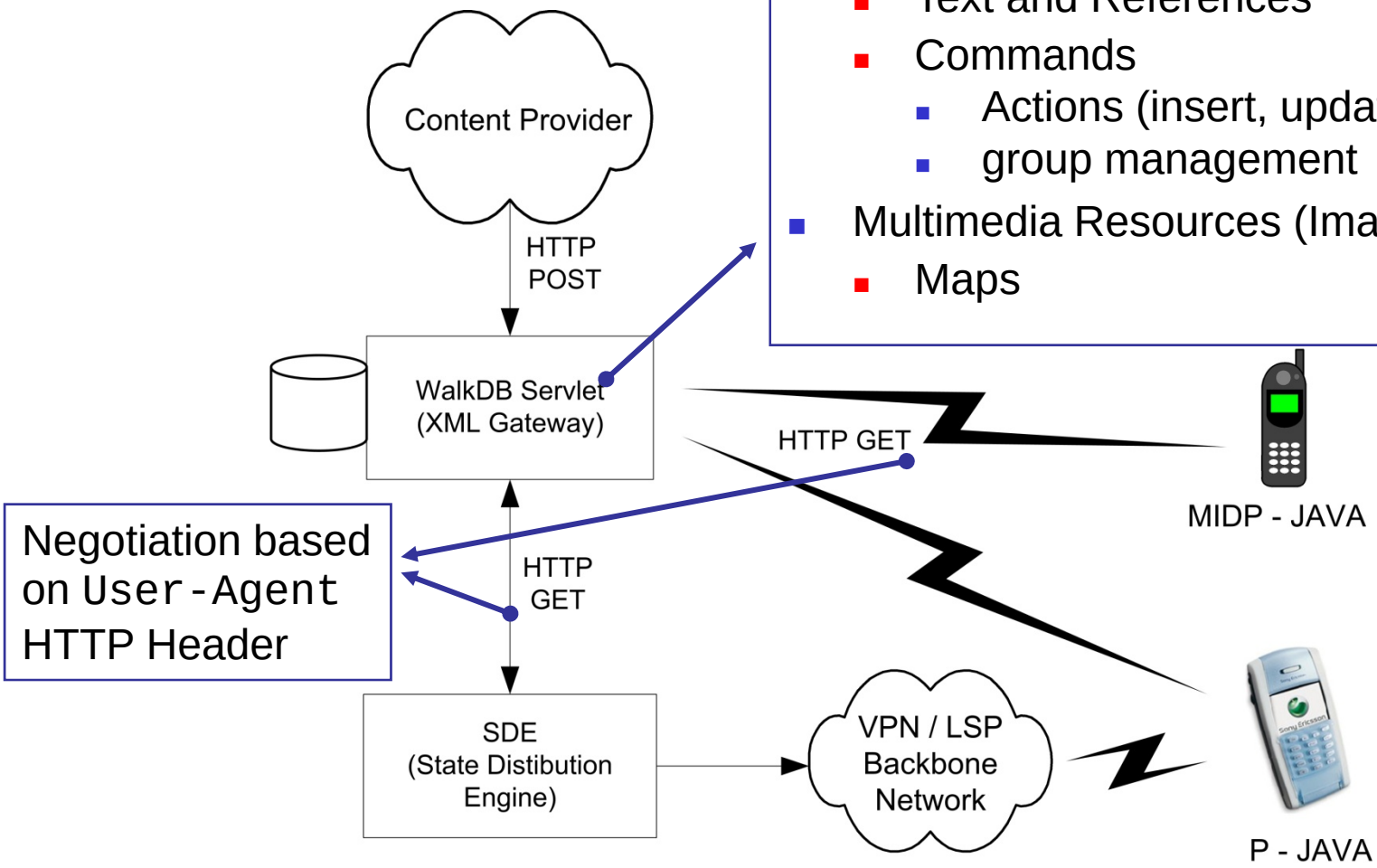
Complex Data Example: CityWalk Dataflow



[www.heywow.com]

Content Adaptation Example: CityWalk

- Adaptation of XML data
  - Text and References
  - Commands
    - Actions (insert, update, delete etc.)
    - group management
- Multimedia Resources (Images, Audio, …)
  - Maps




```
User-Agent: SonyEricssonP800/R101 Profile/MIDP-1.0 Configuration/CLDC-1.0
  DisplayCaps/208x203x12 HeywowClient/Highlander-0.9
```

Parsing Complex Data

- CityWalk Schema Example

```
…
<xsd:complexType name="ElementBaseInfoType">
  <xsd:sequence>
    <xsd:element name="city" type="xsd:string" minOccurs="0" />
    <xsd:element name="name" type="citywalk:LanguageStringType" minOccurs="0" />
    <xsd:element name="description" type="citywalk:LanguageStringType" minOccurs="0" />
    <xsd:element name="mapref" type="citywalk:MapRefType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ElementInfoType">
  <xsd:complexContent>
    <xsd:extension base="citywalk:ElementBaseInfoType">
      <xsd:sequence>
        <xsd:element name="imageResource" type="citywalk:ImageResourceType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="note" type="citywalk:LanguageStringType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
…
```

Parsing Complex Data

- CityWalk Instance Example

```
…
<CTarget Id="LANDSBERG_20030717151816_47B13FF4157213BFC1256D660046FB0F"
    packagePath="dlr.tourGuide.tourGuideContent.">
  <elementInfo>
    <city>Landsberg</city>
    <name>Schmalzturm</name>
    <description xml:lang="DE">
      Auch "Schöner Turm" genannt, gehört zum 1. Mauerring
      aus dem 13. Jahrhundert
    </description>
    <mapref mapId="LandsbergMap1-5" fromLeft="138" fromTop="183" />
    <mapref mapId="LandsbergMap1-0" fromLeft="338" fromTop="383" />
    <imageResource xml:lang="DE"
      ref="schmalzturm.jpg">Schmalzturm</imageResource>
  </elementInfo>
</CTarget>
…
```

XML as serialization format

- Trick: consider XML instances as serialized objects
- If schema is specified carefully (self-describing!), objects (in MIDP Java) can be de-serialized from XML
- Recap: The idea of self-describing serialization is to encode information about class where the object is an instance of, versioning information, length in byte array representation etc.