Working with XML

Hans C. Arents

senior IT market analyst

I.T. Works

"Guiding the IT Professional"

Innovation Center, Technologiepark 3, B-9052 Gent (Belgium), Tel: +32 (0)9 241 56 21 - Fax: +32 (0)9 241 56 56

E-mail: hca@itworks.be - Site: http://www.itworks.be/ - Home: http://www.arents.be/

Working with XML

- Part 1: **Processing and programming XML** 13u30 – 15u15

  coffee break

- Part 2: Storing and querying XML 15u45 – 17u30

  questions & answers

Processing and programming XML

Hans C. Arents

senior IT market analyst

I.T. Works

"Guiding the IT Professional"

Innovation Center, Technologiepark 3, B-9052 Gent (Belgium), Tel: +32 (0)9 241 56 21 - Fax: +32 (0)9 241 56 56

E-mail: hca@itworks.be - Site: http://www.itworks.be/ - Home: http://www.arents.be/

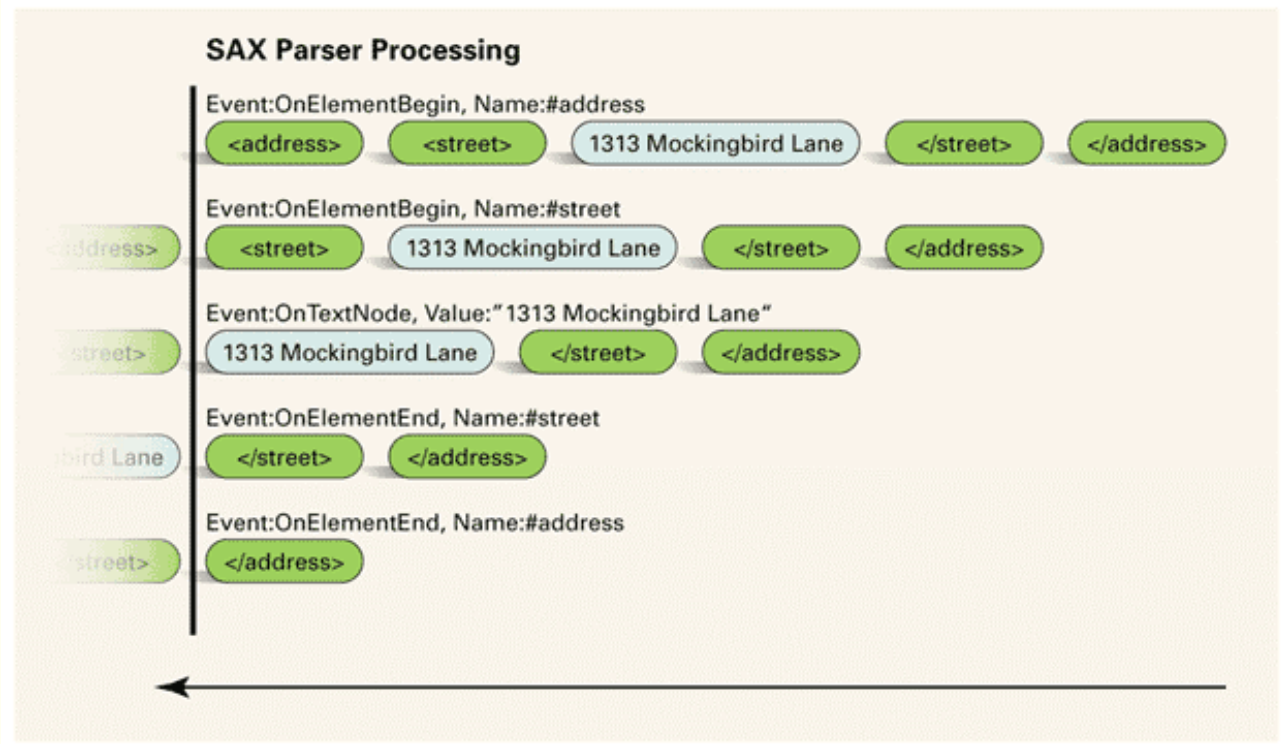
Processing and programming XML

- Why parse?
- Understanding SAX
  - the concepts behind SAX
  - SAX fundamentals
- Understanding the DOM
  - the concepts behind the DOM
  - DOM fundamentals
- Using the DOM in the browser
  - manipulating XML documents using the DOM
  - examples using Microsoft Internet Explorer
- XML parsers
  - on the Microsoft (.NET) platform
  - on the Java platform
  - parser generators

Parsing XML

- XML is an inert document/data format
  - no hard-wired interpretation of the meaning of XML tags, but application-defined interpretation
  - **è** needs to be parsed to become available for processing
- Parsing XML:
  - custom parser
    - write your own parsing code, exposing your own API
  - standard parser
    - use somebody else's parser, but exposing a standard API
- Two standard ways of processing XML:
  - event-based: SAX (**S**imple **A**PI for **X**ML)
    - SAX1, SAX 2
  - tree-based: DOM (**D**ocument **O**bject **M**odel)
    - (Level 0), Level 1, Level 2, Level3

The SAX way of handling an XML document



- Event-based API: SAX (**S**imple **A**PI for **X**ML)
  - uses *callbacks* to report parsing events to the application
  - application deals with these events through customized *event handlers*

  è use for: - batch processing, retrieving data
  - when the parse tree does not have to manipulated

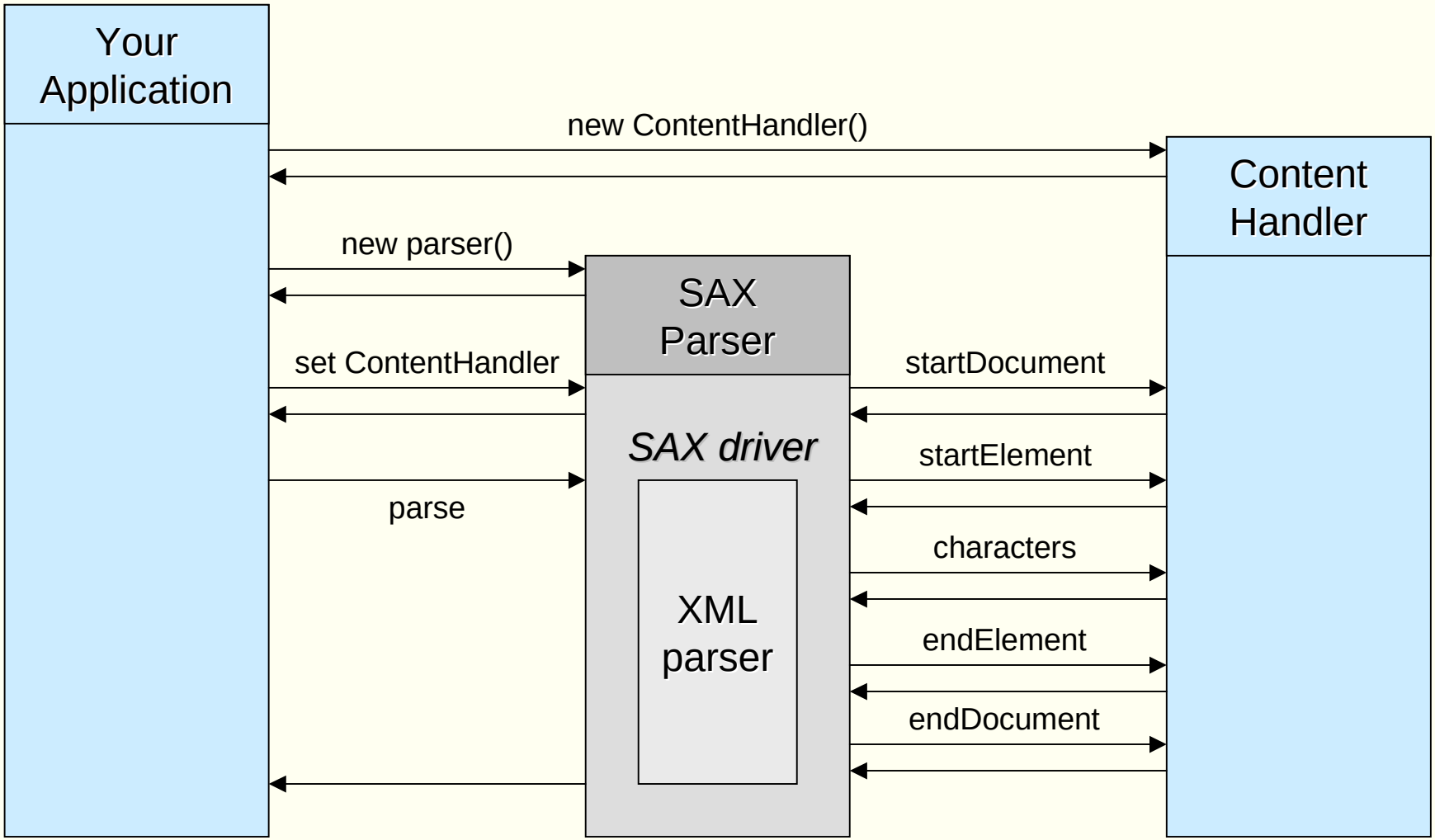
The SAX way of handling an XML document

- Benefits:
  - ü simple and fast
  - ü can parse files of any size
    - memory use is a lot smaller than DOM, does not increase with file size
  - ü useful when you want to build you own data structure
    - create a high-level application-oriented data structure instead of low-level generic DOM tree structure (but watch out for memory use!)
  - ü useful when you only want a small subset of the document
- Drawbacks:
  - û no random access to the document contents
    - problem when the document contains a lot of ID/IDREF cross-references
  - û complex searches can be difficult to implement
    - you have to build the data structures to retain desired context information
  - û no lexical information is made available to the program
    - nothing about the comments in the document, the order of attributes, etc.

What's in SAX?

- Not a W3C standard
  - public domain, developed on **`xml-dev`** mailing list
    `http://www.lists.ic.ac.uk/hypermail-archive/xml-dev/`
  - maintained by David Megginson
    `http://www.megginson.com/SAX/`
- SAX is
  - a set of interface definitions
  - a specification as to how an event-based parser should behave (but parser independent; programs can plug in different parsers)
  - SAX2 adds namespace support, optional validation, optional lexical events for comments, CDATA sections, entity references
- A SAX parser
  - provides concrete class and method definitions for the methods declared in the various SAX interfaces
  - how those interfaces are implemented is up to the parser designer

The SAX parsing process



The SAX parsing process

- SAX1:
  - use the factory method **`ParserFactory.makeParser()`** to retrieve a parser-specific implementation of the **`Parser`** interface
  - your code registers a **`DocumentHandler`** with the parser
  - as the document is read, the parser calls back to the methods of the **`DocumentHandler`** to tell it what it's seeing in the document
- SAX1 callbacks:

```
public void startDocument()
  // receive notification of the start of a document
public void endDocument()
  // receive notification of the end of a document
public void startElement(String name, Attributes atts)
  // receive notification of the start of an element
public void endElement(String name)
  // receive notification of the end of an element
public void characters(char[] ch, int start, int length)
  // receive notification of character data
public void processingInstruction(String target, String data)
  // receive notification of a processing instruction
...
```

The SAX parsing process

- SAX2:
  - use the factory method **`XMLReaderFactory.createXMLReader()`** to retrieve a parser-specific implementation of the **`XMLReader`** interface
  - your code registers a **`ContentHandler`** with the parser
  - as the document is read, the parser calls back to the methods of the **`ContentHandler`** to tell it what it's seeing in the document
- SAX2 callbacks:

```
public void startDocument()
  // receive notification of the start of a document
public void endDocument()
  // receive notification of the end of a document
public void startElement(String namespaceURI, String localName,
                         String qName, Attributes atts)
  // receive notification of the start of an element
public void endElement(String namespaceURI, String localName,
                       String qName)
  // receive notification of the end of an element
public void characters(char[] ch, int start, int length)
  // receive notification of character data
public void processingInstruction(String target, String data)
  // receive notification of a processing instruction
...
```

Creating a SAX2 parser

- The **`XMLReaderFactory.createXMLReader()`** method instantiates an **`XMLReader`** subclass named by the **`org.xml.sax.driver`** system property

```
try { XMLReader parser = XMLReaderFactory.createXMLReader(); }
  catch (SAXException e) { System.err.println(e); }
```

- The **`XMLReaderFactory.createXMLReader(String className)`** method instantiates an **`XMLReader`** subclass

```
try { XMLReader parser = XMLReaderFactory.createXMLReader("org.Ã
                         apache.xerces.parsers.SAXParser"); }
  catch (SAXException e) { System.err.println(e); }
```

- Or you can use the constructor in the package-specific class

```
XMLReader parser = new SAXParser();
```

An example of SAX2 processing

- Input:

```
<?xml version="1.0"?>
<poem style="love">
  <line>Roses are red, violets are blue.</line>
  <line>Sugar is sweet, and so are you.</line>
</poem>
```

- Output:

```
Start document
Start element: poem
Attribute: style, Value = love, Type = CDATA
Start element: line
Roses are red, violets are blue.
End element: line
Start element: line
Sugar is sweet, and so are you.
End element: line
End element: poem
End document
```

An example of SAX2 processing

```
import org.apache.xerces.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;

public class SAXPrint implements ContentHandler {

  public static void main(String[] args) {
    // create a parser
    SAXParser parser = new SAXParser();
    // instantiate a content handler
    SAXPrint printout = new SAXPrint();
    // register the content handler
    parser.setContentHandler(printout);
    for (int i = 0; i < args.length; i++) {
      // parse the document to create the events
      try {parser.parse(args[i]);}
        catch (SAXException e) { System.err.println(e); }
        catch (IOException e) { System.err.println(e); }
    }
  } // end main
```

CONTINUED ON NEXT SLIDE

An example of SAX2 processing

```
// handle startDocument event
public void startDocument() throws SAXException {
  System.out.println("Start document");
} // end startDocument

// handle endDocument event
public void endDocument() throws SAXException {
  System.out.println("End document");
} // end endDocument

// handle characters event
public void characters(char[] ch, int start, int length)
throws SAXException {
  System.out.println(new String(ch, start, length));
} // end characters

// handle endElement event
public void endElement(String namespaceURI, String localName,
String qName) throws SAXException {
  System.out.println("End element: " + localName);
} // end endElement
```

CONTINUED ON NEXT SLIDE

An example of SAX2 processing

CONTINUED FROM PREVIOUS SLIDE

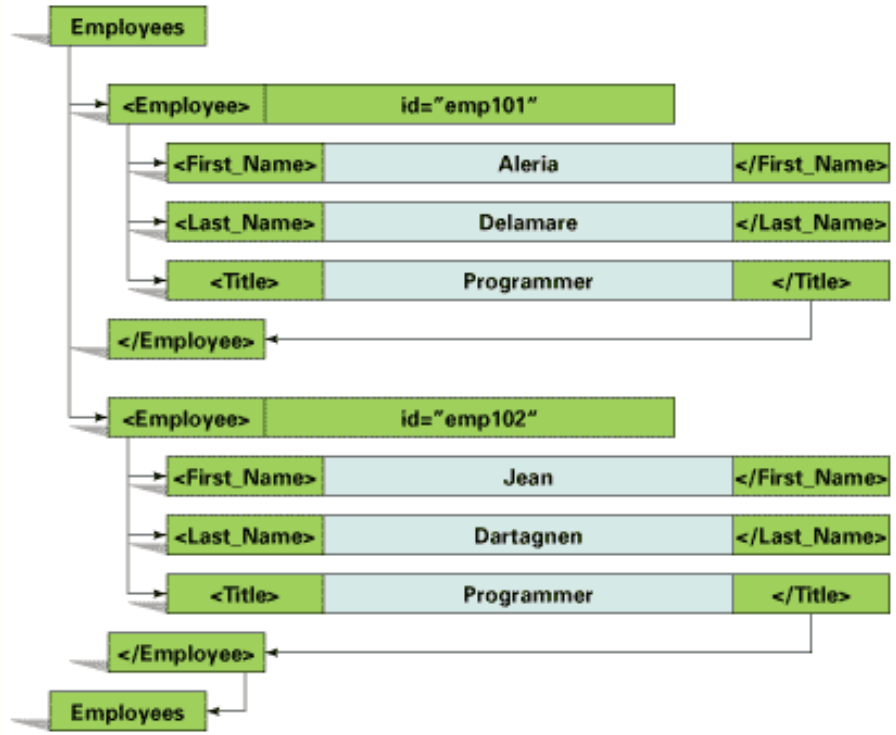
```
  // handle startElement event
  public void startElement(String namespaceURI, String localName,
  String qName, Attributes atts) throws SAXException {
    System.out.println("Start element: " + localName);
    if (atts != null) {
      int len = atts.getLength();
      //process all attributes
      for (int i = 0; i < len; i++) {
        String attName = atts.getName(i);
        String type = atts.getType(i);
        String value = atts.getValue(i);
        System.out.println(
              "Attribute: " + attName
              + ", Value = " + value
              + ", Type = " + type);
      } // end for
    } // end if
  } // end startElement

} // end class SAXPrint
```

Problems with SAX processing

- Read-only
  - for reading XML and extracting data from it, not for modifying it
- Quick-to-forget
  - you do not always have all the information you need at the time of a given callback to the handler
    - e.g. the **`characters()`** method is not guaranteed to give you the maximum number of contiguous characters. It may split a single run of characters over multiple method calls.
  - you may need to store information in various data structures (stacks, queues, vectors, arrays, etc.) and act on it at a later point
- Reusable but not reentrant
  - the same instantiated parser can be reused, without memory penalties
  - but once the parsing process has started, a parser may not be used until the parsing of the document or input has been completed

The DOM way of handling an XML document



- Tree-based API: DOM (**D**ocument **O**bject **M**odel)
  - provides *objects* and *methods* to be used by the application
  - application uses these methods to navigate and manipulate the tree

  è use for: - interactive processing, modifying data
  - adding / modifying / deleting elements and attributes

The DOM way of handling an XML document

- Benefits:
  - ü ensures proper syntax and well-formedness
    - wrong / unclosed / improperly nested tags are automatically avoided
  - ü abstracts data content away from data syntax
    - relational database table, object-oriented database set of objects can present themselves as if they were a DOM tree structure
  - ü simplifies internal document manipulation (add/update/delete nodes)
- Drawbacks:
  - û complex and slow
  - û cannot parse (too) large files
    - a 100 Kb XML file can expand to a 1 Mb DOM tree structure
  - û gets in your way when you want to build you own data structure
    - the low-level generic DOM tree structure has to be disassembled to create a high-level application-oriented data structure
  - û you always have to consider the full document, can't take a subset of it

DOM levels

- (Level 0)
  - what was implemented in Netscape Navigator 3.0 or Microsoft Internet Explorer 3.0 to support DHTML (dialects)
- Level 1
  - access to document contents
  - W3C Recommendation (October 1, 1998)
- Level 2
  - access to document stylesheet + mouse events + traversal
  - W3C Recommendation (November 13, 2000)
- Level 3
  - loading/saving/serializing documents + presentation views + keyboard & device independent events
  - W3C Working Draft (September 1, 2000)

What's in the DOM?

- Document Object Model = tree of nodes
  - node types
    - Document, Element, Attribute, Text, Entity reference
    - Comment, Processing instruction, CDATA section
    - Document type, Entity, Notation, Document fragment
  - each node type has its own properties and methods
    - properties: `firstChild, lastChild, childNodes, parentNode, …`
    - methods: `createElement, createAttribute, createTextNode, …`
- Implementation
  - most complete: Microsoft *MSXML* parser
    - uses DOM node objects, but uses different names and adds its own proprietary objects (useful, but dangerous!)
    - adds its own proprietary properties and methods
      - adding support for DTDs, namespaces, XDR data types and XDR schemas
  - XML parsers for Java (IBM *XML4J*, Apache *Xerces*, …), C++, Python, …

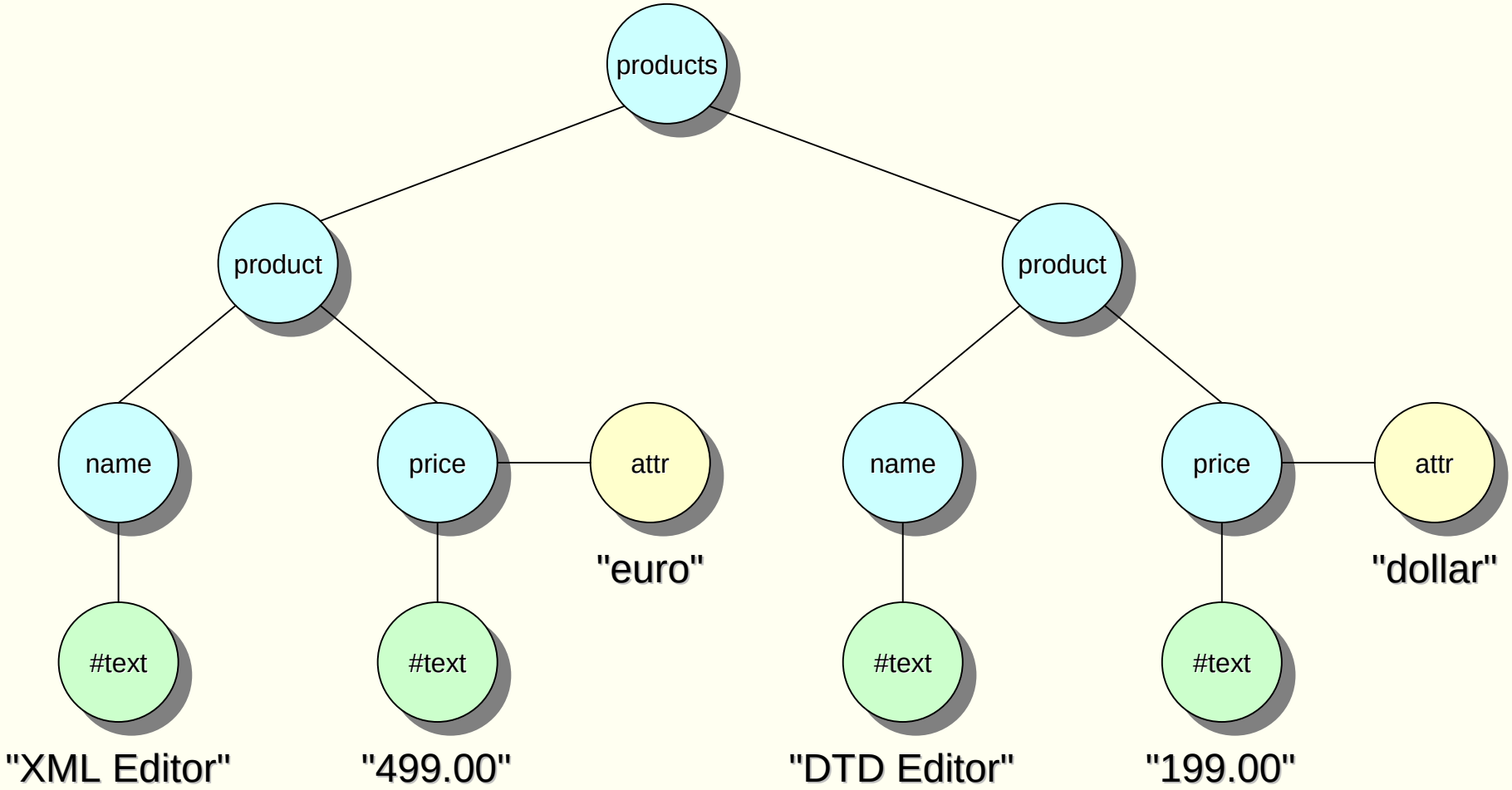
The DOM parsing process

- An XML document:

```
<?xml version="1.0"?>
<products>
  <product>
    <name>XML Editor</name>
    <price currency="euro">499.00</price>
  </product>
  <product>
    <name>DTD Editor</name>
    <price currency="dollar">199.00</price>
  </product>
</products>
```

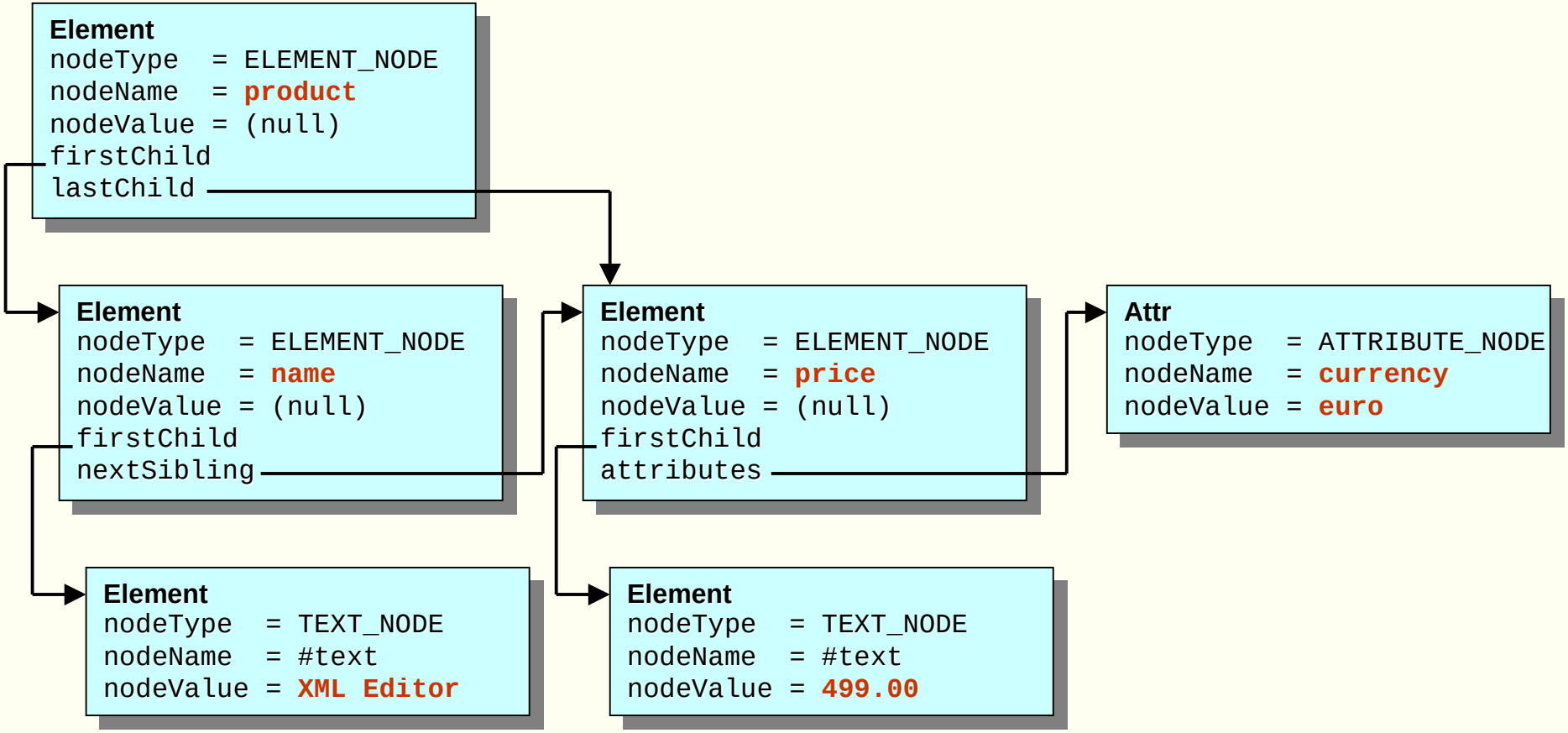
The DOM parsing process

- The resulting DOM tree:



The DOM parsing process

```
<product>
  <name>XML Editor</name>
  <price currency="euro">499.00</price>
</product>
```



DOM objects

- There are 3 fundamental DOM objects:
  - **`Node`** object
    - Document, Element, Attr(ibute), Text, EntityReference,
    - Comment, ProcessingInstruction, CharacterData,
    - Entity, Notation, etc.
  - **`NodeList`** object
    - an ordered collection of Node objects
      - ordered = have a numbered place in a sequence
      - can be accessed by **`index`**
  - **`NamedNodeMap`** object
    - an ordered collection of named Attr(ibute) objects
      - the order is not necessarily the same as in the document
      - can be accessed by **`name`** or **`index`**

DOM properties and methods

- **Node** object properties
  - **childNodes** (returns a **NodeList** object)
  - **attributes** (returns a **NamedNodeMap** object)
  - **firstChild**, **lastChild**, **nextSibling**, **previousSibling**, **parentNode**, etc. (return a **Node** object)
  - **nodeName**, **nodeType**, **nodeValue**, **dataType**, **text**, **xml** (MS only), etc. (return a value)
- **Node** object methods
  - **appendChild**, **replaceChild**, **removeChild**
  - **insertBefore**, **hasChildNodes**
  - **selectNodes**, **selectSingleNode** (MS only)
  - **transformNode** (MS only)

DOM properties and methods

- **`NodeList`** properties and methods
  - **`length`** property
  - **`item(`*`index`*`)`** method
  - **`nextNode`** method (MS only)
- **`NamedNodeMap`** properties and methods
  - **`length`** property
  - **`item(`*`index`*`)`** method
  - **`nextNode`** method (MS only)
  - **`getNamedItem(`*`name`*`)`** method
  - **`setNamedItem(`*`new_node`*`)`** method
  - **`removeNamedItem(`*`name`*`)`** method

DOM properties and methods

- **`Document`** properties and methods
  - represents the whole document and is the only object that allows the creation of other objects (elements, attributes, …)
  - **`createElement`**, **`createTextNode`**, **`createAttribute`**, **`createComment`**, **`createCDATASection`**, etc. methods
  - **`async`**, **`parseError`** properties (MS only)
  - **`load`** method (MS only)
- **`Element`** properties and methods
  - **`tagName`** property, **`getAttribute`**, **`setAttribute`**, **`removeAttribute`**, **`getElementsByTagName`** methods
- **`Text`** and **`CharacterData`** properties and methods
  - **`data`**, **`length`** properties, **`appendData`**, **`deleteData`**, **`insertData`**, **`substringData`** methods

Problems with DOM processing

- Performance
  - building the DOM takes time
    - all processing has to wait until the DOM tree has been built
  - manipulating the DOM takes time
- Memory use
  - the amount of memory used by the DOM is proportional to the size and complexity of the XML document
    - rule of thumb: 1 Kb per node (but there a lot of nodes in the DOM!)
- Generic abstraction
  - the Document Object Model is an (unnecessary?) generic model
    - everything is a node (even if you don't think it should be)
    - language-specific, optimized data structures cannot be used
  - mapping this generic DOM tree structure to your own high-level application-oriented data structure is very inefficient

DOM in Internet Explorer: XML Data Islands

- XML Data Island = XML handling *inside* Internet Explorer 5.x
  - an XML document that exists within an HTML page
  - created using the **<XML>** element within the HTML document
    - in-line XML data

```
<XML ID="XMLID">
  <xmldata>
    <data>text</data>
  </xmldata>
</XML>
```

    - from a file with XML data

```
<XML ID="XMLID" SRC="http://www.site.com/xmlFile.xml"></XML>
```

  - content of the XML Data Island is exposed as a DOM tree

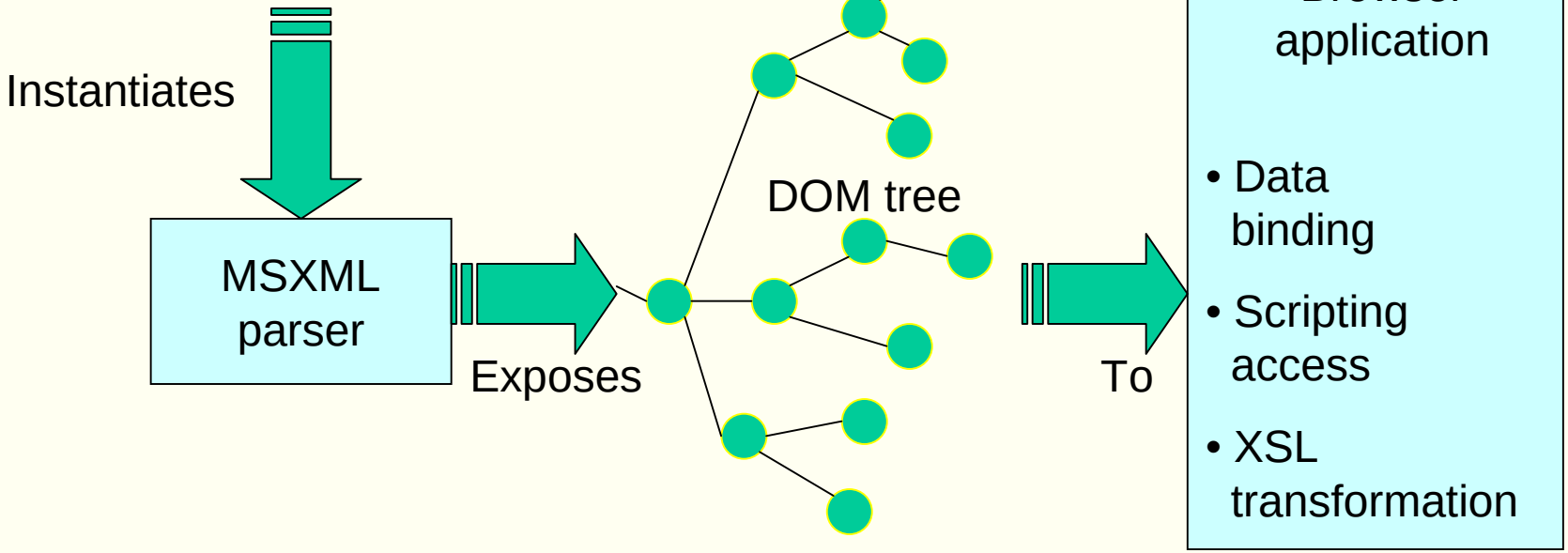
```
function returnXMLDOMTree(){ return XMLID.XMLDocument; }
```

    - to be displayed using the databinding controls of the XML DSO
    - to be accessed from JavaScript or VBScript code
    - to be transformed in-place using XSL

DOM in Internet Explorer: XML Data Islands

```
<XML id="AuthorInfo">
<author>
  <name>Hans Arents</name>
  <affiliation>I.T. Works</affiliation>
</author>
</XML>
```

XML Data Island



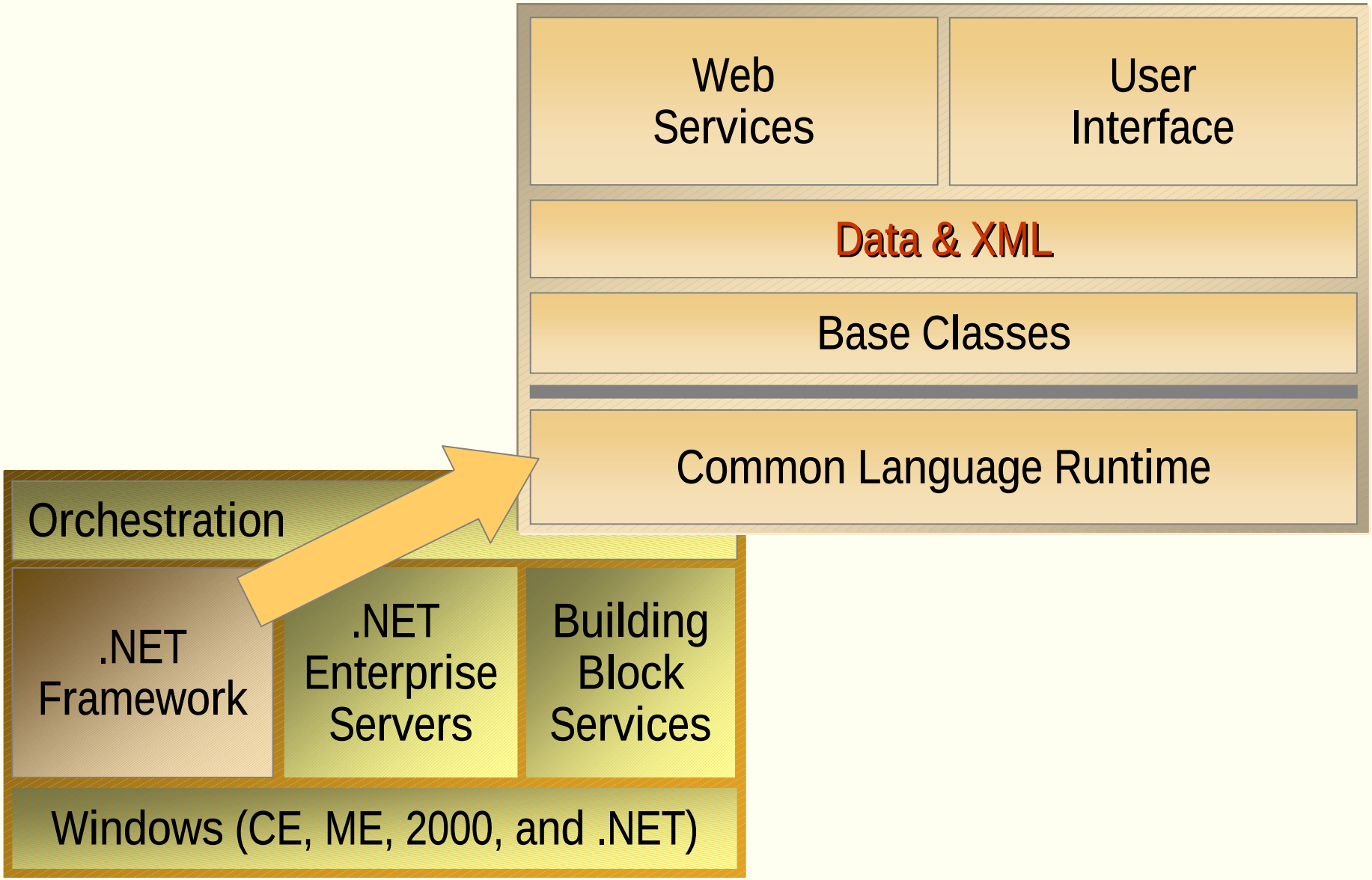
XML parsing on the Microsoft platform

- XML parser is a part of the Windows operating system:
  - Microsoft *MSXML Version 2.5*
    - built-into IE 5.x, shipped with Windows ME / 2000
    - warning: supports pre-standard version of XSL (“MS-XSL”)
  - Microsoft *MSXML Version 3.0* (October 2000)
    - supports XDR schemas, not XML Schemas
    - completely supports standard XSLT and XPath
    - user-defined extension functions in VBScript or JScript
    - optimizations for improved document throughput (2/3x faster)
      - e.g. server-side XSL stylesheet caching
    - can be installed alongside or as a replacement of the original *MSXML*
    - COM object accessible from C++, Visual Basic and scripting environments
  - Microsoft *MSXML Version 4.0* (somewhere in 2001) for use in .NET
    - support for XML Schemas, XML query
- **http://www.netcrucible.com/xslt/msxml-faq.htm**

Features of the MSXML parser

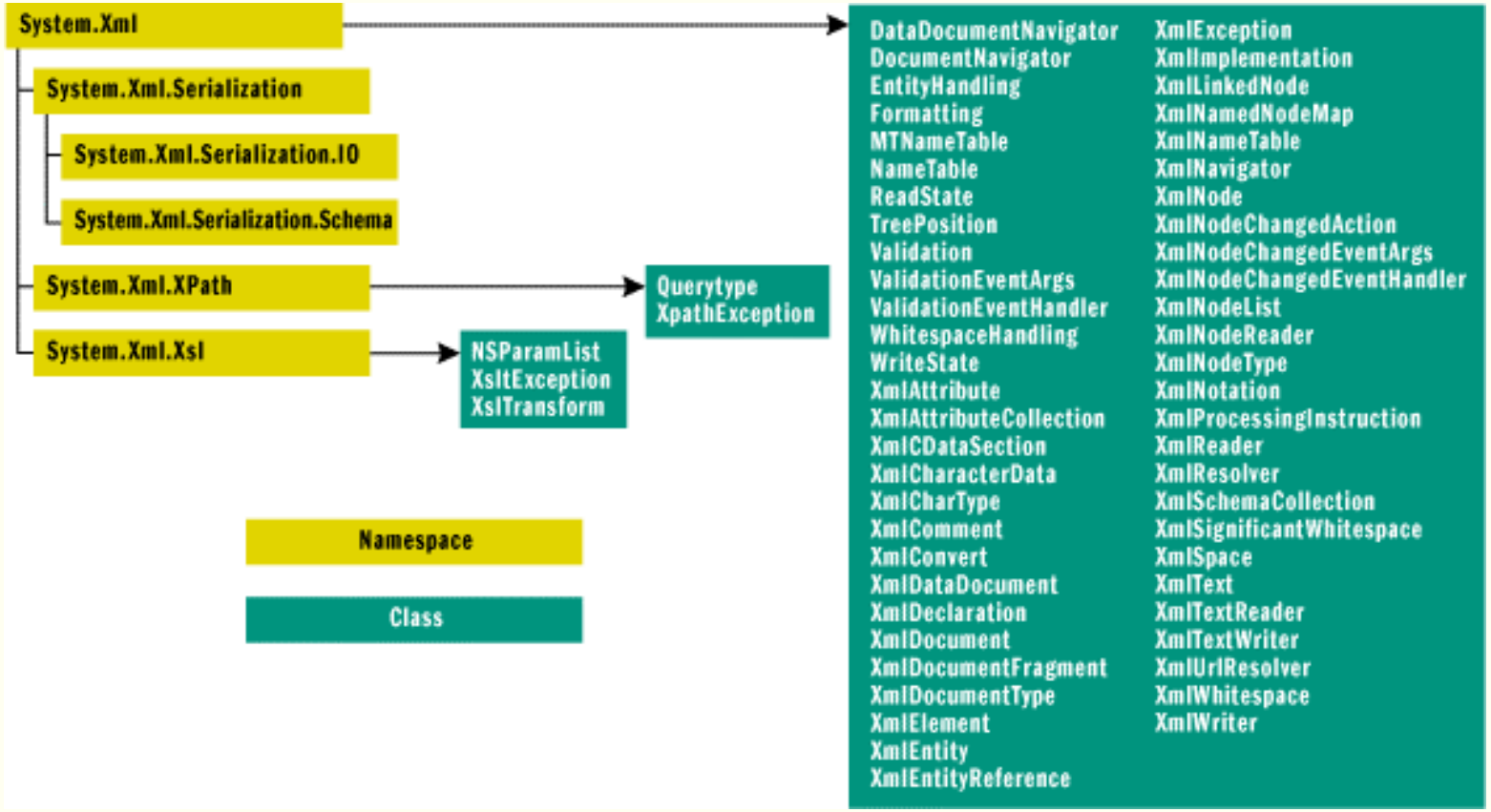
- Two ways of interacting with XML using MSXML parser:
  - DOM (**D**ocument **O**bject **M**odel):
    - load XML data and build an in-memory tree with *objects* and *methods* to be used by the application
      - navigating, querying and modifying through a standardized API
      - Microsoft-specific extensions: **á** development ease, but **â** portability
    - in *MSXML 2.5*: DOM Level 1, in *MSXML 4.0*: DOM Level 2
  - SAX (**S**imple **A**PI for **X**ML):
    - read XML data and generate *callbacks* to be treated by *event handlers*
    - in *MSXML 2.5*: SAX 1, in *MSXML 3.0*: SAX 2
- Validation of XML documents/data against:
  - DTD: in *MSXML 2.5*
  - + XDR (**X**ML **D**ata-**R**educed): in *MSXML 3.0*
  - + XSD (**X**ML **S**chema **D**efinition Language): in *MSXML 4.0*

The . Net Framework



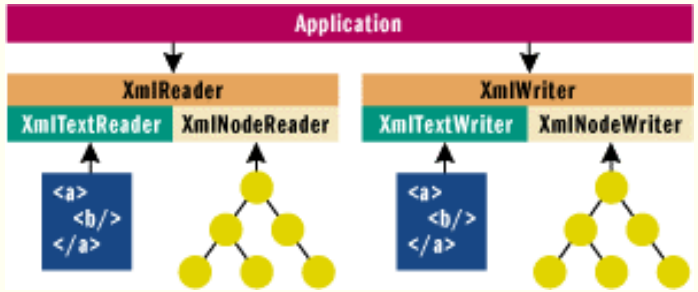
XML support in the .NET Framework

- .NET Framework classes
  - core technology substrate of .NET
  - used by ASP+, ADO+, Web services, …



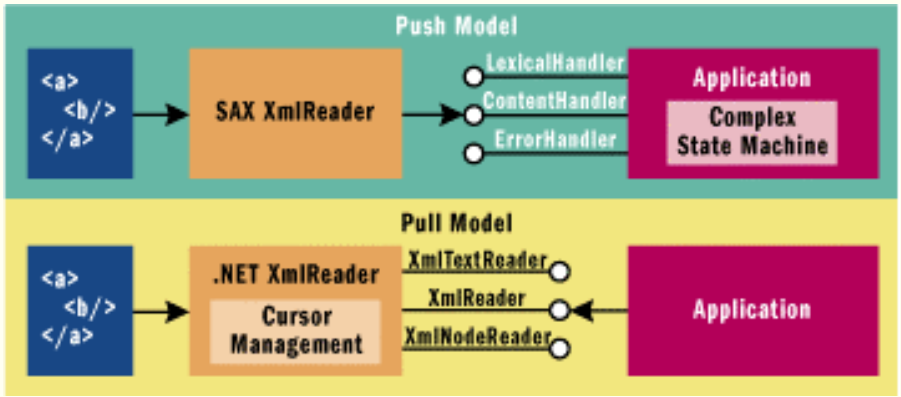
XML support in the .NET Framework

- Two abstract base classes for the handling of streaming XML:
  - XmlReader:
    - for fast, forward-only, read-only processing of an XML stream
    - contains 3 concrete derived classes:
      - XmlTextReader, XmlNodeReader and XslReader
  - XmlWriter:
    - for producing XML streams that conform to XML 1.0 + Namespaces
    - contains 2 concrete derived classes:
      - XmlTextWriter and XmlNodeWriter
- A developer uses:
  - **XmlNodeReader** and **XmlNodeWriter**: for working with in-memory DOM trees
  - **XmlTextReader** and **XmlTextWriter**: for reading from/writing to text-based stream
  - custom Reader and Writers (adapted to his/her specific needs)



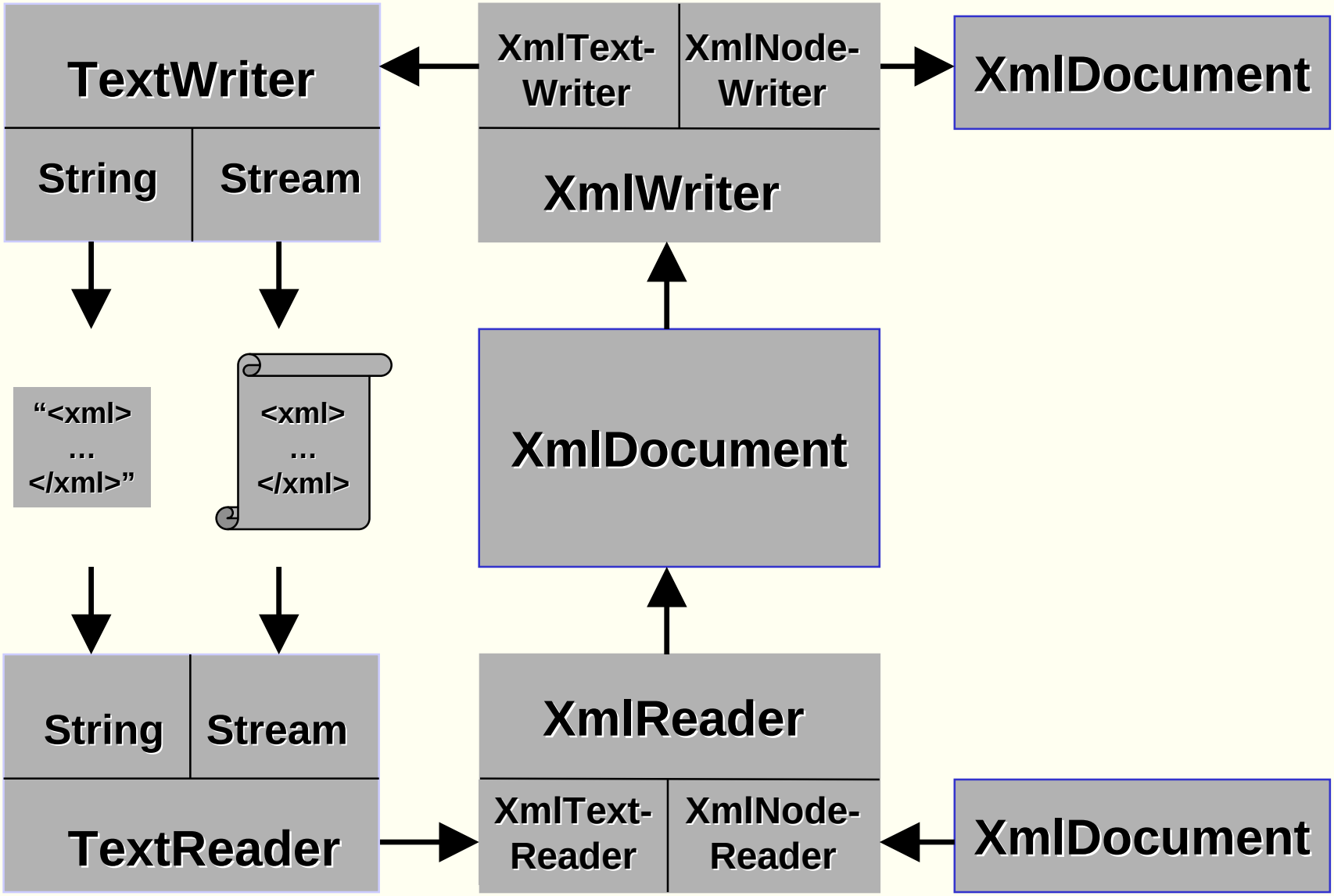
XML support in the .NET Framework

- XmlReader is a compromise between DOM (simple XML programming model) and SAX (efficient XML data processing)
  - not *push*: all elements have to be handled one by one
  - but *pull*: loop through elements, skip unwanted elements, …



- DOM support in .NET Framework classes:
  - standard DOM: DOM Level 1, DOM Level 2
  - custom DOM:
    - DOM loading is built on top of XmlReader, DOM serialization on top of XmlWriter **è** you can extend how the DOM interacts with your applications

XML support in the .NET Framework

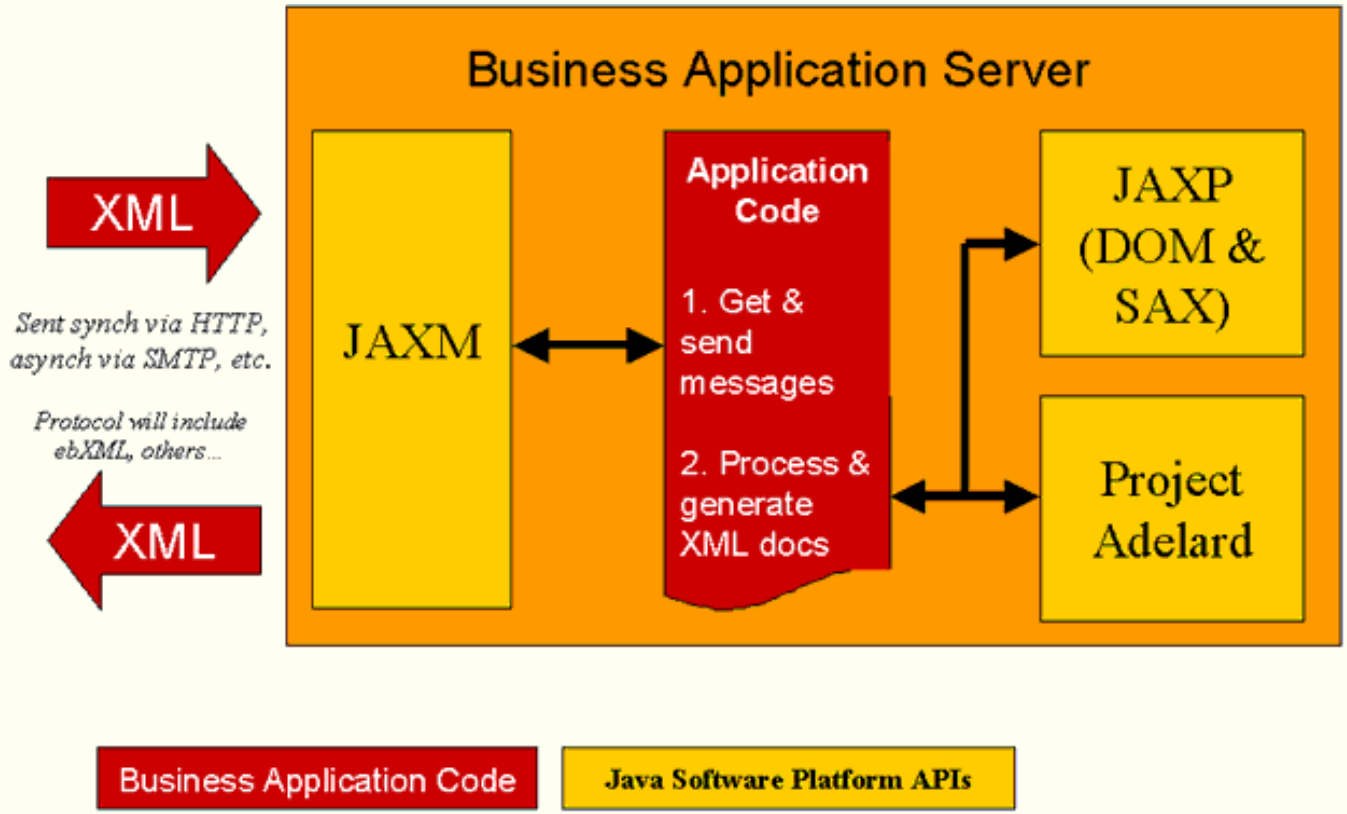


XML parsing on the Java platform

- Not a single XML parser, but a choice of XML parsers

Parser	URL	Validating	Namespaces	DOM1	DOM2	SAX1	SAX2	License
Apache XML Project's Xerces Java Parser 1.3.0	http://xml.apache.org/xerces-j/	X	X	X	X	X	X	Apache Software License, Version 1.1
IBM's XML Parser for Java 3.1.1	http://www.alphaworks.ibm.com/formula/xml	X	X	X	X	X	X	License
Sun's Java API for XML Processing 1.1	http://java.sun.com/xml/download.html	X	X	X	X	X	X	Free
Sun's Java API for XML Processing 1.0	http://java.sun.com/xml/archive.html	X	X	X		X		Free
Oracle's XML Parser for Java 2.1.0 (beta)	http://technet.oracle.com/tech/xml/parser_java2/	X	X	X	X	X	X	Free
Oracle's XML Parser for Java 2.0.2.10	http://technet.oracle.com/tech/xml/parser_java2/	X	X	X		X		Free
James Clark's Expat	http://www.jclark.com/xml/expat.html					X		MIT
James Clark's XP	http://www.jclark.com/xml/xp/					X		Modified BSD

XML handling in Java: core support



- XML extensions of the Java language:
  - JAXP = **J**ava **A**PI for **X**ML **P**arsing (= SAX1 + DOM1 + factory classes)
  - JAXM = **J**ava **A**PI for **X**ML **M**essaging
  - JAXB = **J**ava **A**PI for **X**ML Data **B**inding (Project Adelard)

XML handling in Java: JDOM

- JDOM = **J**ava **D**ocument **O**bject **M**odel
  http://www.jdom.org/
  - not built on or modeled after DOM, but integrates with DOM and SAX
  - open source project with an Apache-style license
  - has been officially accepted as JSR-102
- Goal: represent an XML document for easy and efficient accessing, reading, manipulation and writing
  - straightforward, lightweight & fast API
  - *Java-optimized* to be easy to use by Java programmers
    - use the power of the Java 2 language
    - take advantage of method overloading, the Collections APIs, reflection, weak references, …
    - provide conveniences like built-in type conversions

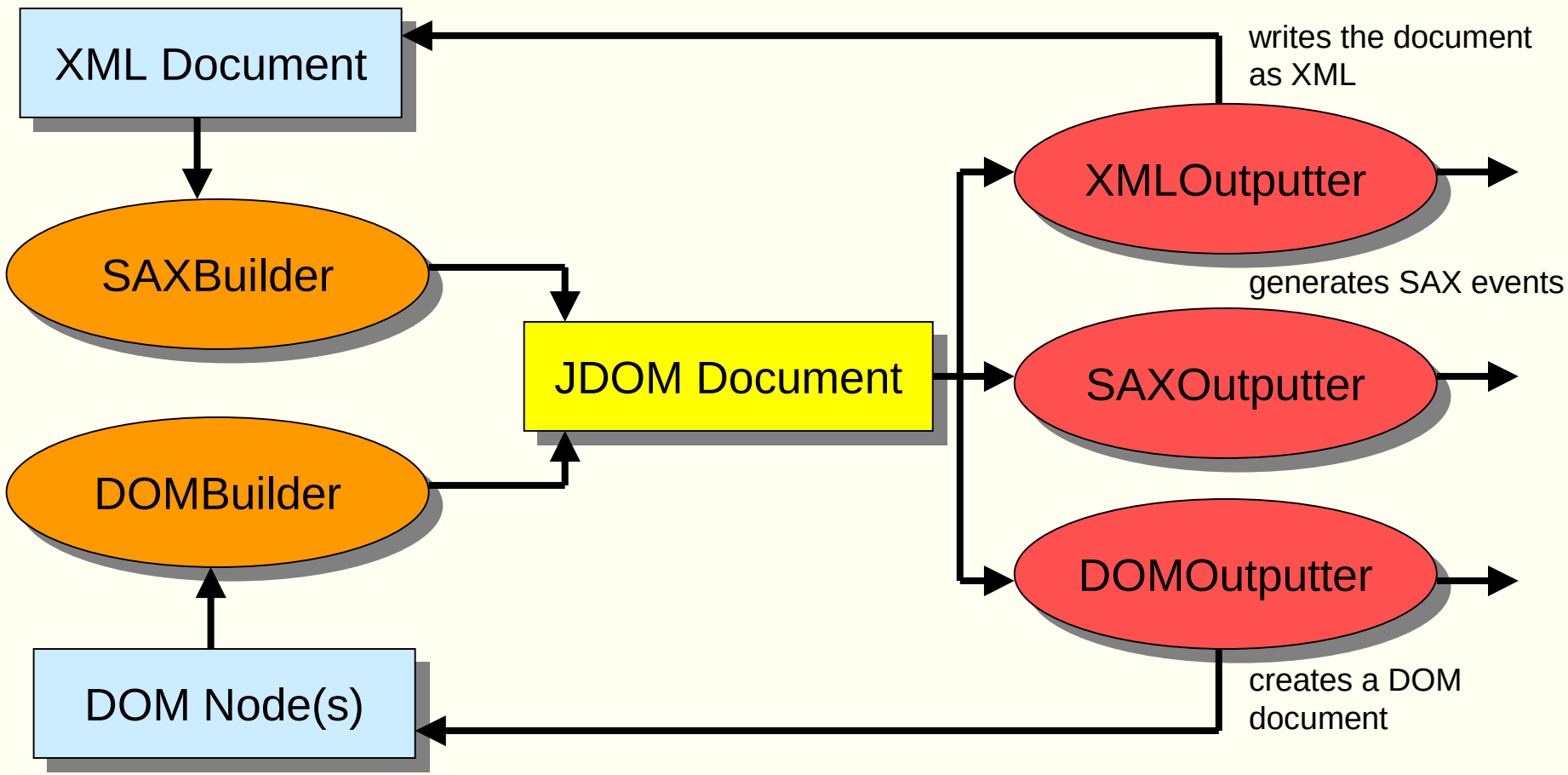
→ to be included as another core XML API in JAXP 1.2 (or 2.0)

The JDOM philosophy

- JDOM should hide the complexities of XML wherever possible
  - an **`Element`** has content, not a child **`Text`** node with content
  - exceptions should contain useful, comprehensible error messages
  - give line numbers and error specifics, use no SAX or DOM specifics
- JDOM should integrate with DOM and SAX
  - support reading and writing DOM documents and SAX events
  - easy conversion from DOM/SAX to JDOM and back
  - support runtime plug-in of *any* DOM or SAX parser
- JDOM should stay current with the latest XML standards
  - DOM Level 2, SAX 2.0, XML Schema
- JDOM does not need to solve every problem
  - it should solve 80% of the problems with 20% of the effort

  è it probably got the ratios to 90% / 10%

General JDOM program flow



JDOM vs. DOM

- Create a simple XML document in JDOM:

```
Document doc = new Document(
  new Element("rootElement")
  .setText("This is a root element"));
```

- Create a simple XML document in DOM:

```
Document myDocument = new org.apache.xerces.dom.DocumentImpl();

 // Create the root node and its text node,
 // using the document as a factory
 Element root = myDocument.createElement("myRootElement");
 Text text = myDocument.createText("This is a root element");

 // Put the nodes into the document tree
 root.appendChild(text);
 myDocument.appendChild(root);
```

JDOM vs. DOM

- Create this XML content:

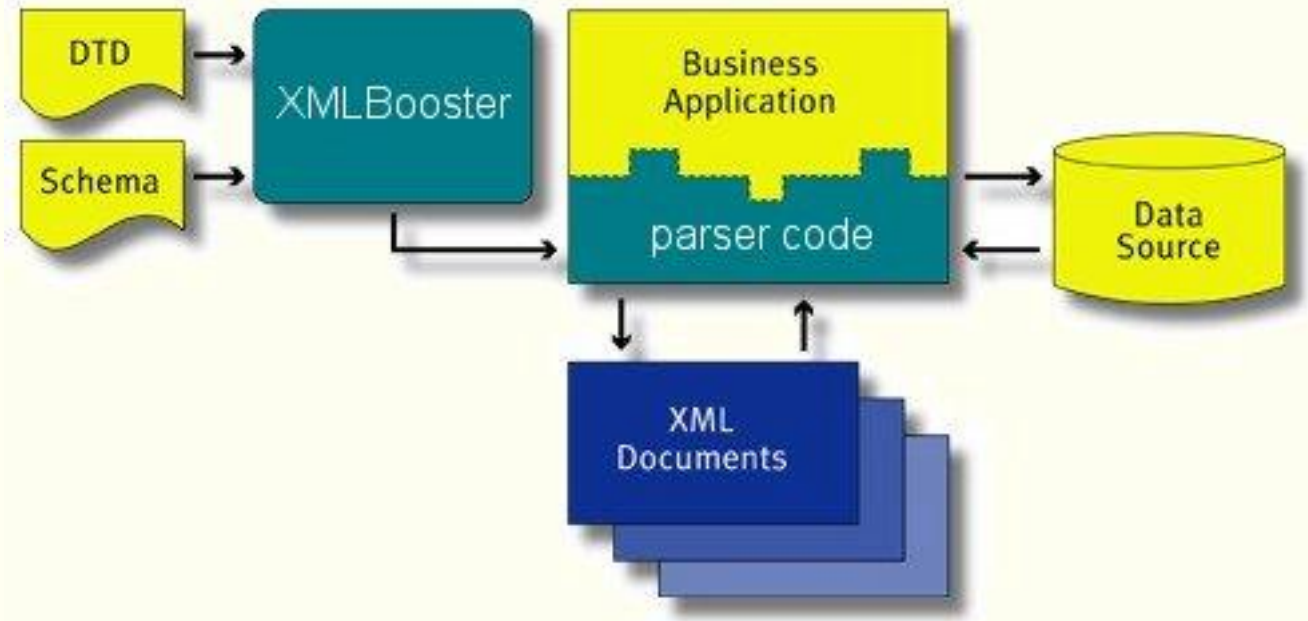
```
<linux-config>
  <gui>
    <window-manager>
      <name>Enlightenment</name>
      <version>0.16.2</version>
    </window-manager>
  </gui>
</linux-config>
```

by using this JDOM code:

```
Document doc = new Document(
  new Element("linux-config")
    .addContent(new Element("gui")
      .addContent(new Element("window-manager")
        .addContent(new Element("name")
              .setText("Enlightenment"))
        .addContent(new Element("version")
              .setText("0.16.2"))
      )
    )
  );
```

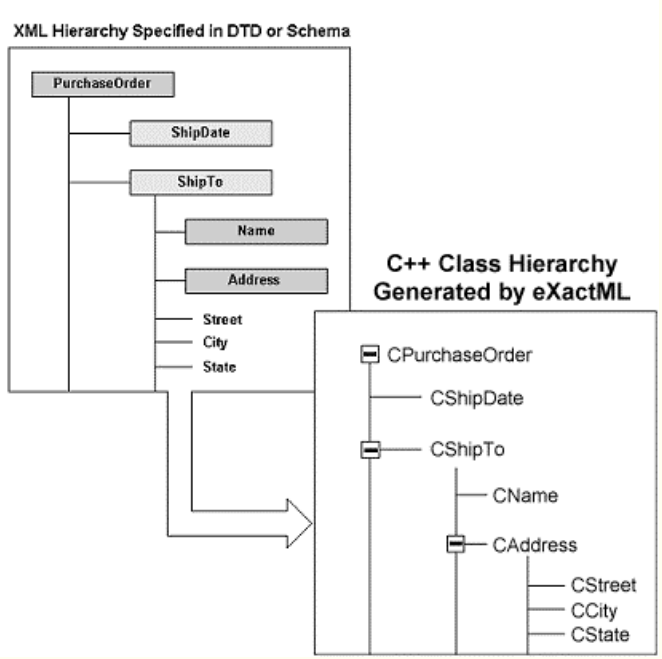
XML parser generators

- Generating custom parsers
  - parser code optimized for a specific DTD/XML Schema
  - e.g. *XMLBooster* (http://www.xmlbooster.com/)
    - turns DTD into Java, C++, C, Delphi, COBOL parser code
      - using native data structures rather than a poorly typed generic DOM tree
    - can produce parsers with finer-grained validation than offered by an DTD by using a custom schema language



XML parser generators

- Generating custom code
  - native code hiding the details of how to handle XML data
    - *marshalling*: class **à** XML / *unmarshalling*: XML **à** class
    - use classes directly instead of SAX/DOM
    - check well-formedness and validity
    - serialize / deserialize XML data
  - e.g. *eXactML* (**http://www.bristol.com/**)
    - turns DTD / XML Schema into C++ classes
    - turns XML data structures into C++ objects



- XML marshalling is being built into all major programming languages
  - Java, Microsoft C#, …

Selecting the right XML tool

- Take your time for the tool selection
  - Due diligence vs. “Wow, this is neat!”
    - run the product vs. look at the demo
  - Does it work? Does it *really* work?
    - does it fully comply with the … standard?
  - Does it work with your other tools?
    - with the XML editor/parser/processor/… you’ve chosen
  - What happens under peak conditions?
    - with big XML files? with large volumes? at very high load? …
- Select the best tools you can afford
  - Use off-the-shelf components whenever possible
    - look at open source software if you can
  - Spend energy solving your business problem, not fighting the tools!

XML tools sites

- The *XML Resource Guide* at *XML.com*
  - **http://www.xmltools.com/**
- Collection of XML tool descriptions
  - **http://www.xmlsoftware.com/**
- The *XMLTree* XML directory
  - **http://www.xmltree.com/**