Contributions to Information Integration from XML Technology

Mario Jeckle
DaimlerChrysler Research and Technology Ulm
mario.jeckle@daimlerchrysler.com
mario@jeckle.de
www.jeckle.de
Table of Contents

Contributions to Information Integration from XML Technology

- Mediators
- Wrappers for Extracting Data
- Communications
- XML Data Binding
- Web Services
- Efficient and Flexible Queries
- XQuery
- Security Aspects
- XML Digital Signatures
- XML Encryption
- W3C’s XML Schema
- Wrappers for Extracting Data
- XSLT
Outline of a XML-based Information Infrastructure

System

Data

in-memory objects

XML Data Binding

Document Object Model

Simple API for XML

W3C's XML Schema

<?xml ...>

<?xml ...>

<?xml ...>

<?xml ...>

<?xml ...>

<?xml ...>

<?xml ...>

<?xml ...>

<?xml ...>

<?xml ...>

Application Programmer's Interface

f_1(x)

f_2(x)

f_3(x)

f_4(x)

f_5(x)

f_n(x)

... XML Encryption

XML Digital Signature

M. Jeckle – Contributions to Information Integration from XML Technology

2002-05-02 3
Modeling the Contents of an Information System

Motivation:

- XML inherited the Document Type Definition (DTD) from SGML
- The DTD mechanism ...
  - was designed to support document oriented structuring
  - defines a strictly hierarchical information structure
    (i.e. a tree containing elements as inner nodes and
    all the information placed within text and attribute nodes as leaves)
  - offers a rich set of string-based types :-(
  - does not support the new XML secondary standards like
    Namespaces or the Information Set
  - is not deployed (re-)using XML
  - does not come with a metamodel
- XML in contrast to SGML is used for expressing
  data not documents

=> DTDs cannot be considered as sufficient for XML
Modeling the Contents of an Information System

Purpose:

- XML schema ...
  - defines a definition language for XML vocabularies
  - offers facilities for describing structure and constraining content of XML v1.0 documents
  - allows documents to exploit XML's newer features like Namespaces
  - Reconstructs DTD's capabilities and hence forms a superset of the expressive power found there
  - Allows meta-validation (i.e. the validation of the schema itself)
Modeling the Contents of an Information System

Basic Idea:
• W3C's XML schema is ...
  • a grammar for formulating arbitrary XML vocabularies
  • The specification consists of ...
    • Part 0 providing some introduction and basic examples
    • Part 1 describing structures and content of a XML vocabulary
    • Part 2 describing datatypes used within part 1 and elsewhere
  • a significant improvement over DTD's expressive power
    (in the long term it will practically supersede it)
  • a XML language itself
    (i.e. it consists of a vocabulary accompanied with a defined semantics)
  • a combination all of the major competing successors
  • a W3C recommendation since 2001-05-02
  • the basis of all W3C's second generation standards
    (e.g. XPath v2.0, XSLT v2.0, XHTML v2.0, XMLP/ SOAP, ...)
• well supported by tools
Modeling the Contents of an Information System

Technology: • XML Schema ...
  • Enhances structural capabilities, e.g. by the introduction of re-useable structured types
  • adopts the idea of meta modeling
  • removes constructs with unclear semantics (e.g. Entities and Notations)
  • comes with an extended and extensible typesystem
Modeling the Contents of an Information System

Technology: • XML Schema ...
- Enhances structural capabilities, e.g. by the introduction of re-useable structured types
- adopts the idea of meta modeling
- removes constructs with unclear semantics (e.g. Entities and Notations)
- comes with an extended and extensible typesystem
Wrappers for Extracting Data

Motivation:
• XML-encoded data is inherently offline, i.e. it is persistently stored in files or virtual files (a.k.a. XML database)
• To "execute" (i.e. create, render, or process) XML it must be loaded into memory
• writing parsers for specific XML vocabularies (e.g. XHTML, XML Schema, SVG) seems to be perfectly superfluous ...

Purpose:
• Since XML forms a meta-language and generic XML parsers are available there must be a generic API for processing XML within applications

Basic Ideas:
1. Orientate control flow along XML's structural primitives (i.e. element, attribute, etc.)
2. Create a generic data structure capable of representing all of XML's structural primitives in memory
3. Create a generic transformation capable of transferring the specific structures of an arbitrary XML vocabulary into problem specific memory structures
Wrappers for Extracting Data

Technology:

- XML Data Binding
- Document Object Model
- Simple API for XML
Wrappers for Extracting Data

Technology:

- XML Data Binding
- Document Object Model
- Simple API for XML

- SAX is ...
  - not a parser (but, it can be used to construct one)
  - a collection of programming language neutral interfaces
  - Available in various implementations (e.g. C(++), Python, Perl, Eiffel, Java ...)
  - an generic event-based push-model API realized with call-back functions
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42" />
</root>
```

```java
... public void startDocument()
{ ... }
public void startElement(...) 
{ ... }
public void endDocument()
{ ... }
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
</root>
```

```java
... public void startDocument()
{ ... }
public void startElement(...)
{ ... }
public void endDocument()
{ ... }
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:

```
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
</root>
```

```
... public void startDocument()
  { ... }
public void startElement(...)
  { ... }
public void endDocument()
  { ... }
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
</root>
```
```java
... public void startDocument()
  {... }
public void startElement(...)
  {... }
public void endDocument()
  {... }
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
</root>
```
```java
... public void startDocument()
  {... }
public void startElement(...)
  {... }
public void endDocument()
  {... }
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Pros and Cons:
- **Pros:**
  - Minimal constant memory consumption
  - As generic as it gets
- **Cons:**
  - Impossible (i.e. quite difficult to code) to track state
  - No modifications possible
  - Memory representation is only built-up transiently and is lost after method completion
Wrappers for Extracting Data

Technology:

- XML Data Binding
- Document Object Model
- Simple API for XML

• DOM is ...
  • not a parser (but, it can be used to construct one)
  • programming language neutral, but prerequisites an object-oriented system
  • an abstract tree-based in-memory representation of a XML document
  • capable of reading in, modifying, and writing back XML structures
  • available in various implementations (e.g. C++, Java, Eiffel, ...)
**Wrappers for Extracting Data**

**Technology:**
- XML Data Binding
- Document Object Model
- Simple API for XML

**Example:**
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
    <foo bar="42"/>
    <baz>xyz</baz>
</root>
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```
**Wrappers for Extracting Data**

**Technology:**
- XML Data Binding
- Document Object Model
- Simple API for XML

**Example:**
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```

M. Jeckle – Contributions to Information Integration from XML Technology
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Pros and Cons:
- Pros:
  - In-memory representation of a whole XML document which allows arbitrary modifications
  - Still generic, w.r.t. specific XML vocabularies
- Cons:
  - Could become quite memory consuming
  - Costly navigation and information access
  - Often still too generic...
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

XML Data Binding is ...
- not a parser (but, it could be used to construct one, even if I would not recommend it)
- programming language bound, in fact the idea came from the Java community
- an algorithm for transferring arbitrary XML vocabularies into Java class structures
- capable of reading in, modifying, and writing back XML structures
- available only in Java implementations so far
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Basic Idea:

XML Schema

<table>
<thead>
<tr>
<th>XML Schema</th>
<th>Schema Compiler</th>
<th>.java</th>
<th>Java Compiler (e.g. javac)</th>
<th>.class</th>
</tr>
</thead>
</table>

M. Jeckle – Contributions to Information Integration from XML Technology
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:

```xml
<schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <element name="baz" type="xs:string"/>
  <element name="foo">
    <complexType>
      <attribute name="bar" type="xs:integer" use="required"/>
    </complexType>
  </element>
  <element name="root">
    <complexType>
      <sequence>
        <element name="foo" ref="foo"/>
        <element name="baz" ref="baz"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

M. Jeckle – Contributions to Information Integration from XML Technology
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Example:
```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="baz" type="xs:string"/>
  <xs:element name="foo">
    <xs:complexType>
      <xs:attribute name="bar" type="xs:integer" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="foo" ref="foo"/>
        <xs:element name="baz" ref="baz"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Wrappers for Extracting Data

Technology:

- XML Data Binding
- Document Object Model
- Simple API for XML

Example:

```xml
<schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="baz" type="xs:string"/>
  <xs:element name="foo">
    <xs:complexType>
      <xs:attribute name="bar" type="xs:integer" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="foo" ref="foo"/>
        <xs:element name="baz" ref="baz"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</schema>
```
Wrappers for Extracting Data

Technology:
- XML Data Binding
- Document Object Model
- Simple API for XML

Pros and Cons:
- **Pros:**
  - Most appropriate way to transfer specific XML vocabularies into in-memory representations
  - No genericity left (no overhead in processing)
- **Cons:**
  - No genericity left (needs re-built cycle every time schema changes)
  - Only available for the Java programming language
Mediators (for Transferring Data)

Motivation:
- (Prerequisite: XML is everywhere and becomes as the ASCII of the 21st century ubiquitous)
- XML and XML Schema make it quite easy to create new vocabularies
- The internet makes it quite easy to spread them
- Since XML is a meta-format it must be possible to establish a meta-transformation mechanism

Purpose:
- Transferring XML streams into XML streams (it is also possible to create arbitrary Unicode sequences)

Technology:
- Orientated on well known concepts, such as functional programming
- Provides an easy to use (XML-) language to encode these transformations
- Relies on the existence of generic transformation engines
- Provides a Turing-complete XML-style programming language
Mediators (for Transferring Data)

Example:

```
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```

```
<?xml version="1.0"?>
<foo>
  <bar>42
     <baz content="xyz"/>
  </bar>
</foo>
```
Mediators (for Transferring Data)

Example:

```xml
<?xml version="1.0"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="root">
    <foo>
      <xsl:apply-templates select="foo"/>
    </foo>
  </xsl:template>
  <xsl:template match="foo">
    <bar>
      <xsl:value-of select="@bar"/>
      <xsl:apply-templates select="//baz"/>
    </bar>
  </xsl:template>
  <xsl:template match="baz">
    <baz content="{.}"/>
  </xsl:template>
  <xsl:template match="text()"/>
</xsl:transform>
```
Mediators (for Transferring Data)

Example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="root">
    <foo>
      <xsl:apply-templates select="foo"/>
    </foo>
  </xsl:template>
  <xsl:template match="foo">
    <bar>
      <xsl:value-of select="@bar"/>
      <xsl:apply-templates select="//baz"/>
    </bar>
  </xsl:template>
  <xsl:template match="baz">
    <baz content="{.}"/>
  </xsl:template>
  <xsl:template match="text()"/>
</xsl:transform>
```
Mediators (for Transferring Data)

Example:
```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <foo bar="42"/>
  <baz>xyz</baz>
</root>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="root">
    <foo>
      <xsl:apply-templates select="foo"/>
    </foo>
  </xsl:template>
  <xsl:template match="foo">
    <bar>
      <xsl:value-of select="@bar"/>
      <xsl:apply-templates select="//baz"/>
    </bar>
  </xsl:template>
  <xsl:template match="baz">
    <baz content="{.}"/>
  </xsl:template>
  <xsl:template match="text()"/>
</xsl:transform>
```
Communication -- Web Services

Characteristic Attributes of Web Services:

• **Immaterial**
  Services do not produce physical goods

• **Transient**
  Services cannot be stored to consume them at a later point in time

• **Location Dependant**
  Services cannot be transferred to another location

• **Production Consumption Synchronicity**
  Service provider and service consumer cannot be decoupled
Communication -- Web Services

Increasing Interoperability

Decreasing Proprietary

- UDDI
- DISCO
- ADS
- Naming service
- ...
Communication -- Web Services

VB C++ C# ... Java ... ...

Common Language Specification
Java Bytecode

Web Services

WSDL UDDI

SOAP

XML Schema
XML 1.0 + Namespaces

HTTP, SMTP, ...
TCP/IP
Netzwerk
Communication -- Web Services

A Web Service is a piece of server-site software that provides a certain functionality (as a black box) and is accessible through Internet protocols using XML/SOAP messages with a described and published interface (typically by means of WSDL). Those interface descriptions should be registered in a (global) registry like UDDI.
Communication -- Web Services

- VB
- C++
- C#
- ...
- Java
- ...
- ...

Common Language Specification
Java Bytecode

Security
Session handling

WSDL
UDDI
SOAP
XML Schema
XML 1.0 + Namespaces
HTTP, SMTP, ...
TCP/IP
Netzwerk
Securing Web Services

Application Layer
- XML Application Layer

Presentation Layer (SOAP Layer)
- SOAP
  - BEEP
  - SMTP
  - HTTP
  - MIME
  - SSL

Session Layer (wire protocol)
- TCP
  - UDP

Transport Layer
- IP, X.25, SPX, IPX

Network Layer
- HDLC, SLIP, PPP, Ethernet, IEEE 802.x

Data Link Layer
- STP, UTP, V. 90, ISDN, DSL

Physical Layer
Securing Web Services -- XML Digital Signatures

Purpose: Modular, application-level signature mechanism
- Data Integrity
- Non-Repudiation
- Authorization
- Authentication

Basic Idea:

Technology:
- Message Digest, one-way-function
- Cryptographic signature of message digest
- Scalable

Standards:
- XML-Signature Syntax and Processing (W3C Recommendation since 2002-02-12 also IETF RFC 3072)
Securing Web Services -- XML Encryption

Purpose: Modular, application-level encryption mechanism
• Confidentiality
• Authorization

Basic Idea:

Technology: • Cryptographic encryption algorithms
• Symmetric and asymmetric encryption
• Key management needed
• Interoperable with digital signatures

Standards: • XML Encryption Syntax and Processing
(W3C Candidate Recommendation since 2002-03-04)
Securing Web Services -- Securing Web Content

Purpose: Application-transparent content encryption
• Confidentiality
• Data Integrity
• Non-Repudiation
• Authorization
• Authentication

Basic Idea: Adaptive security negotiation
• Certificate exchange
• Combines signature and encryption mechanisms

Technology: The TLS Protocol Version 1.0 (RFC 2246)
• The Secure HyperText Transfer Protocol (RFC 2660)
• Netscape's SSL / HTTPS

Standards: XML, SOAP, HTTP, BEEP, SMTP, FTP, TCP, UDP, IP, X.25, SPX, IPX, HDLC, SLIP, Ethernet, PPP, IEEE 802.3, STP, UTP, ISDN, DSL, Modem
**Securing Web Services -- Securing Network Content**

**Purpose:** Session-transparent content encryption  
- Confidentiality  
- Data Integrity  
- Non-Repudiation  
- Authorization  
- Authentication

**Basic Idea:**

**Technology:**  
- Operates solely on IP Layer  
- Allows encryption of payload and/ or packet headers  
- Two-way authentication via key management protocol

**Standards:**  
- Security Architecture for IP (RFC 2401, 1998-11)
Efficient and Flexible Queries

Motivation:
• More and more data is stored using XML …
  • in files
  • in so-called "XML databases" (a.k.a. virtual documents)
• Considering XPath a query language could be quite harmful …
  • as a locator language it actually does not extract any data
  • it's calculus-based expressions are pretty handy for smaller queries, but only for those …
• Additionally, there are extensions to XPath needed, which are not standardized among vendors

=> What we need is an SQL for XML

Purpose:
• Creating and extracting XML data primarily from XML data sources, but not limited to them
Efficient and Flexible Queries

History:

- XPath
- XPointer
- XSL-Patterns
- Lorel
- XML-QL
- XQL
- XQL-99
- SQL
- W3C's XQuery
- OQL
- YATL
- UnQL
- XSL-Patterns
- Quilt
Efficient and Flexible Queries

Syntax:

**FLWR:**
- **forClause**: for → Variable → in → Exp → Variable
- **letClause**: let → Variable → := → Exp → Variable
- **whereClause**: where → Exp

**Exp:** extended XPath
Efficient and Flexible Queries

Example:

FOR $persons IN //Person
LET $projects := //Project
WHERE $projects/@ProjectLeader = $persons/@PersID
RETURN $persons/Name
Efficient and Flexible Queries

Example:

FOR $persons IN //Person
LET $projects := //Project
WHERE $projects/@ProjectLeader = $persons/@PersID
RETURN $persons/Name
Efficient and Flexible Queries

Example:

FOR $persons IN //Person
LET $projects := //Project
WHERE $projects/@ProjectLeader = $persons/@PersID
RETURN $persons/Name
Efficient and Flexible Queries

Example:

```xml
FOR $persons IN //Person
LET $projects := //Project
WHERE $projects/@ProjectLeader = $persons/@PersID
RETURN $persons/Name
```