Syntax

XML Schema

XML Techniques for E-Commerce, Budapest 2004

Institute for Information Processing and Microprocessor Technology (FIM)
Johannes Kepler University Linz, Austria

E-Mail: sonntag@fim.uni-linz.ac.at
http://www.fim.uni-linz.ac.at/staff/sonntag.htm

© Michael Sonntag 2004
Questions?

Please ask immediately!
• Introduction
  → Why?
• Schemas: The BIG solution (or isn’t it?)
  → All problems solved?
• Simple types
  → Sequences of elements
• Complex types
  → Attributes, choices of elements, character content
• Datatypes
  → Defining the character content in more detail
XML Techniques for E-Commerce: Syntax

XML Schema

- ebXML, SOAP, Security, Metadata, ...
- XSLT
- HTML
- XML
- FO
- Java
- XPath
- XML Name-space
Why bother with a structural / syntactical description?

- Suppose, companies C (consumer) and P (producer) want to do business with each other
  - C sends to P: Order
  - P sends to C: Error "Delivery date is missing"
  - C sends to P: "Didn't know you need it. Here is it: '25.6.2004' ".
  - P sends to C: Error "Delivery date is invalid: MM/DD/YY"
  - C sends to P: "We will let you know: '06/25/04' "
- A lot of time, work and nerves were necessary!
- Solution: A "contract" between both parties
  - Both agree before, exactly what data to exchange
    » Syntax and semantics
    » Syntax: Schema
    » Semantics: DAML+OIL, OWL, ...
Why bother with a structural / syntactical description?

- Resulting process between C (consumer) and P (producer)
  - C creates an order for P
  - C verifies it according to the contract (schema)
  - C changes it until it matches the contract
  - C sends it to P
  - P verifies the order according to the contract (schema)
  - P processes the order

- Barring rare problems (changes during transmission, etc.) only one exchange takes place and all verification of input is done in one step
  - Subsequent programs need not check the input anymore!
    » For syntax; semantics checks are still needed!
Why Schemas?
Isn't DTD good enough?

- DTD cannot be reused (short of physical copying)
- No object orientation: Deriving, etc.
- Datatypes are extremely weak
  - Only for attributes, not for content
  - Content validation therefore sorely lacking
- DTD itself is not XML
  - Cannot be created/handled by XML-applications
  - Requires separate modules, ...
- Only simple constraints
- No namespaces available

- Advantages of DTD: Easy to learn, simple
A new scheme: Schemas

- Allows definition of structures
  - Complex types: May contain elements and attributes (and text)
  - Simple types: Text-only content
  - Element, attribute declarations: Similar to DTDs
- Allows definition of datatypes
  - Predefined: Date, time, boolean, numbers, strings, …
  - Extensions possible and recommended!
  - Examples: nonNegativeInteger, unsignedByte, hexBinary, string,…
- Contains “element substitution groups”
  - Substituting one named element for another
  - Allows a kind of inheritance
  - Extensions or restrictions from base elements
- Rather complicated, but VERY powerful!
Validity of XML

- Applies to XML if DTD or schemas are used
- Well-formed documents need not adhere to a / their structure specification
  - If they do, they are „valid“ (the same as validity with DTD's)
- Schemas: Many additional rules
  - E. g. content must match the specified datatype

- Validity checking verifies the syntax of the document on a higher level
  - „Basic“ syntax: Well-formedness (correct forming & tagnames & nesting of tags)
  - „Extended“ syntax: Validity (correct names & correct content)
    » Schemas: Correct datatype, ...
The schema as a whole

- The "base" element of an schema
  → Need not be the document element; could be embedded

- Two namespaces must be distinguished
  → The namespace of the schema itself (example below: "xs")
    » E. g. "element" to be used to define tags
  → The namespace to be defined for the content
    (example below: Unnamed = no prefix)
    » E. g. "element" to be used as a tag

- Example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.kingsgoods.com/freeshop">
  <xs:element name="element" type="xs:string"/>
</xs:schema>
```

Schema namespace

Defining an element

What to define

(Unnamed) namespace to be specified
Schemas and namespaces

- The namespace to be defined is called "target namespace" 
  - Defined within the root element
    - Attention: Local declarations are NOT within this namespace!
    - Only global ones (=top-level, directly beneath the document element) and those explicitly defined (by using the prefix)
  - Can be omitted: Unnamed namespaces (usually a very bad idea!)
- To make sure everything always (even when included somewhere) works fine:
  - Set `elementFormDefault="qualified"` and `attributeFormDefault="qualified"`
  - All local elements in instance documents MUST be qualified
    - This can be replaced with defaulting
  - All local attributes in instance document MUST be qualified
    - No defaulting for attributes: Must always be prefixed!
  - Global definitions within the schema must always be prefixed when used within the schema itself!
**Example:**

\[
\text{\textless ?xml version="1.0" encoding="UTF-8"?\textgreater} \\
<x:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" \\
targetNamespace="http://www.fim.uni-linz.ac.at/Emerald/Konverter/ConfigurationFile" \\
xmils:cf="http://www.fim.uni-linz.ac.at/Emerald/Konverter/ConfigurationFile" \\
elementFormDefault="qualified" attributeFormDefault="qualified"> \\
\]

**Instance document:**

\[
\text{\textless ?xml version="1.0" encoding="UTF-8"?\textgreater} \\
<cf:CPS-Converter-Configuration cf:configVersion="1.1" \\
xmils:cf="http://www.fim.uni-linz.ac.at/Emerald/Konverter/ConfigurationFile" \\
xmils:xsi="http://www.w3.org/2001/XMLSchema-instance" \\
xsi:schemaLocation="http://www.fim.uni-linz.ac.at/Emerald/Konverter/ConfigurationFile Schemata/ConfigurationFile.xsd"> \\
<cf:taxonomyFiltering cf:filteringEnabled="false">… \\
\]
---

**From DTD to XSD**

- A simple XML document
- and it's DTD

compared to

- The same document referencing a schema
- and it's schema

Using a schema in an instance document:

- Reference the schema-instance namespace
  - `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

- Use the location attribute from this namespace to specify where to find the definition(s) for the namespace(s)
Schemas: Simple types

- For constraining character content of elements
  - Cannot contain other elements; only text
- A specification of a kind of "datatype"
  - See later!
- Example:
  - `<xs:simpleType name="celsiusWaterTemp">
    `<xs:restriction base="xs:number">
      `<xs:fractionDigits value="2"/>
      `<xs:minExclusive value="0.00"/>
      `<xs:maxExclusive value="100.00"/>
    `<xs:restriction>
  `<xs:simpleType>
  - Defines a type for the temperature of liquid water: Is a number with two fractional digits between 0 and 100 (both exclusive)
  - Range: 0.01 - 99.99
Schemas: Attribute declaration

- Specifying the content of the attribute
  - Each attribute must have a name and a type
    » Type: Simple datatype (see above); no complex types allowed!
- Optional or required
  - Optional: Default for attributes
    » Specify attribute "use" as "optional" if desired to make explicit
  - Required:
    » Specify attribute "use" as "required"
- Default or fixed; both together is not possible!
  - Default values:
    » Use attribute "default"; always optional
  - Fixed values:
    » Use attribute "fixed"; can be optional or required
Examples:

- `<xs:attribute name="lastCleaned" type="xs:date" use="required"/>`  
  This attribute can be added to any element and has a content of the type "date" (a basic datatype). Must be present if added to an element definition.

- `<attribute name="hatSize" type="xs:string" default="medium">`  
  `<annotation><xs:documentation>Can be either small, medium, or large</xs:documentation></annotation>`
  This attribute is of type string, has a default value (and might therefore be omitted), and some explanations (which will NOT show up on actual use of the attribute; see later!); The restrictions on the content value therein will not be enforced!

- `<xs:attribute name="shirtsize" type="xs:string" use="required" fixed="XXXL"/>`  
  This attribute must always have the value "XXXL". Still, it must always be specified explicitly (even though parsers could supply it automatically!)

[Attributes.xsd, Attributes.xml]
Schemas: Complex types

- Specifying the allowed attributes, child elements, text content
- Controls derivation and substitution
- Can be abstract or final
- May be „mixed“ (elements and character data) or „elements only“
- Can be named or anonymous
  - If named they can be reused, e.g. for derivation
- Type for defining structured content
  - Simple types: Character content only!
  - Complex types: Child elements, attributes and character content!
Example (details will come later!):

- `<xs:complexType name="apartmentType">
  <xs:sequence>
    <xs:element name="address" type="InternationalAddress"/>
    <xs:element name="owner" type="PersonType"/>
    <xs:element ref="comment" minOccurs="0"/>
  </xs:sequence>
  <xs:choice>
    <xs:element name="house" type="houseType"/>
    <xs:element name="flat" type="flatType"/>
  </xs:choice>
  <xs:attribute name="lastCleaned" type="xs:date"/>
</xs:complexType>

- An apartment has an address and an owner and perhaps a comment. Additionally the type of the house or the type of the flat (depending on what it is) must be specified. It must contain an attribute when it was last cleaned.
Complex types: Simple vs. complex content

- Attention: Different from simple vs. complex type!
- A complex type may contain either simple or complex content (but only exactly one type)
- Simple content:
  - Can only contain character data
  - Cannot contain elements
  - Difference to simple type: Can contain attributes!
  - Can use "extension", but not "restriction"
- Complex content:
  - Can contain character data, elements and attributes
  - Can be defined as empty (no data, no characters; perhaps attributes)
  - Usually used as a sort of "derivation" from a base type
Complex types: Adding attributes

- Simple types cannot have attributes
  - Use complex type to combine them
- `<xs:complexType name="price">`
  `<xs:simpleContent>`
  `<xs:extension base="xs:decimal">`
  `<xsd:attribute name="currency" type="xsd:string"/>`
  `</xs:extension>`
  `</xs:simpleContent>`
  `</xs:complexType>`

- Result:
  `<xs:element name="Price" type="price"/>`
  `<price currency="EUR">123.5</price>`

- Attributes and attribute groups must always be at the end of a complex type definition!

SimpleContent.xsd, SimpleContent.xml
Schemas: Element declaration

- Allows specification of
  - Name: The local name (=tag) of the element
  - Namespace: Which NS this element will be part of
  - Datatype for content: Specifies the allowed content
  - Occurrence constraints: How often it can/must appear
  - Whether it is abstract: Cannot be used, only its substitutions
  - Annotation: Human- and machine-targeted metadata
  - Reference: Not defined here, but somewhere else using a name
  - Default value: Default content if empty (see also later)
  - Final: What extensions/restriction are allowed
  - Fixed: Fixed content
  - ...
Schemas: Element declaration

● Example definition:
  → `<xs:element name="myHome" type="tns:apartmentType"/>
     » The name of the element is "myHome"
     » It will be of type "apartmentType", which must be defined somewhere
       – See example above for this type!
       – Usually a global declaration in this file

● Example usage in instance document:
  → `<myHome lastCleaned="01.04.2002">
      <address/></owner/>
      <house/>
    </myHome>
    // comment is optional
    // OR "flat", but not both!

● Example usage within the schema
  → `<xs:element ref="myHome"/>
      Element.xsd, Element.xml"
Complex types: Occurrence constraints

- Number of times an element may occur can be specified:
  - minOccurs: Must occur this times at least
    - Range: 0…N
  - maxOccurs: Can occur this times at most
    - Range: 0…N, "unbounded"
  - Default value for both is "1": Must occur exactly once

- Only for elements!
  - Attributes: Use the "use" attribute in specification

- Example:
  - `<xs:element name="partyMember" type="xs:integer" minOccurs="2" maxOccurs="unbounded"/>
    - A party consists of at least 2 persons and has no upper limit
  - `<xs:element name="giftWrapping" ... minOccurs="0" maxOccurs="1"/>
    - Gift wrapping is optional; only one wrapping layer allowed!
Schemas: Four types of complex types

- Elements containing only other elements
  → Complex content
- Empty elements
  → Complex content
- Elements containing both elements and character data
  → Complex content
- Elements containing only character data (text)
  → Simple content
  → Only useful if attributes are needed. Otherwise use a simple type!

- All of them may contain attributes
Complex types: Child elements

- **Choice:** Only one element from the contained ones may actually be present in a document
  - Occurrence constraint: Several of the elements may appear in any order

- **Sequence:** All contained elements must appear in exactly the specified order
  - If number of occurrence of an element is >1, they must appear immediately after each other (before the next element in the sequence)
  - Occurrence constraint: Complete sequence may appear several times

- **All:** Special case, only allowed at the top of a content model
  - No element may appear more than once (only 0 or 1 times is allowed)
  - Elements may appear in any order
  - Usually not a good idea
  - Not described here!

- **Can also contain "model groups":** Similar to attribute groups
  - Not described here!
Complex types: Choice example

→ <xs:complexType name="TreasureObjectType">
   <xs:choice>
     <xs:element name="metal" type="MetalType"/>
     <xs:element name="artobject" type="ArtType"/>
     <xs:element name="preciousstone" type="StoneType"/>
   </xs:choice>
   <xs:attribute name="value" type="xs:float"/>
</xs:complexType>

→ A treasure object is either a metal an art object or a precious stone
   » Problem: Golden statue with emerald eyes?

→ Valid examples:
   » <treasure value="150"><metal name="gold"/></treasure>
   » <treasure value="0.10"><artobject creator="Sonntag"/></treasure>
   » <treasure value="900"><preciousstone type="diamond"/></treasure>

→ Invalid example:
   » <treasure value="280"><metal name="gold"/><artobject creator="Michelangelo"/></treasure>
Complex types:
Choice example

→ `<xs:complexType name="VehicleType">
   <xs:choice minOccurs="1" maxOccurs="2">
     <xsd:element name="truck" type="TruckType"/>
     <xsd:element name="car" type="CarType"/>
   </xsd:choice>
</xsd:complexType>

→ Most vehicles are either cars or trucks, but some of them are classified as both (legally seen)

→ Each car must be of at least one type, some may be of two types
   » Problem: E. g. two times "car" is also allowed! ("<car/><car/>")

→ Valid examples:                        → Invalid examples:
   <vehicle><truck/></vehicle>            <vehicle></vehicle>
   <vehicle><car/></vehicle>             <vehicle><car/> <car/></vehicle>
   <vehicle><car/><truck/></vehicle>     <car/></vehicle>
Complex types:
Sequence example

→ `<xs:complexType name="TreasureType">
    <xs:sequence>
        <xs:element name="owner" type="PersonType"/>
        <xs:element name="container" type="ContainerType"/>
        <xs:element name="content" type="TreasureObjectType"
            minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="value" type="xs:float"/>
    <xs:attribute name="weight" type="xs:float" use="optional"/>
</xs:complexType>

→ A treasure has an owner and is within a container followed by one or more treasure objects. It also has a value and optionally a weight.
    » Only 'content' may appear several times
    » But only grouped together at the end
Complex types: Syntax

Sequence example

→ `<xs:complexType name="PersonType">`  
  `<xs:sequence minOccurs="2" maxOccurs="2">`  
  `<xs:element name="parent" type="PersonType"/>`  
  `<xs:element name="parentRelation" type="RelationType"/>`  
  `</xs:sequence>`  
  `</xs:complexType>`

→ Each person has exactly two parents and a certain relation with them

  » The complete sequence must appear two times
    – OK: `<person><parent><parentRelation><parent><parentRelation></person>`
    – WRONG: `<person><parent><parent><parentRelation><parentRelation></person`

  » Is this really a good design?
    – Better: Put the relation into the parent element and specify occurrence on this element (`<xs:element name="parent" type="PersonType" minOccurs="2" maxOccurs="2"/>`)  
    – How will we actually specify the "roots" of the family tree, where we don't know the parents?
Default values

- Default values for elements:
  - Add attribute "default", its value is the default element content
  - If the element is empty, its (missing) content text is replaced by the default value
  - If the element is missing, nothing happens
    » It is not inserted and filled with the default value!

- Default values for attributes:
  » See above!
  - If an attribute is missing, it is created & filled with the default value
  - If an attribute is empty, it keeps its value
    » It is not filled with the default value!

- Summary of default values:
  - Elements are filled if empty, but not created if missing
  - Attributes are not filled if empty, but created if missing
● `<xs:element name="size" type="xs:string" default="Medium"/>
  (within a "shirt" element)
  → `<size>Medium</size>`
  → `<size>Large</size>`
  → `<shirt>` is NOT added within!

● `<xs:attribute name="country" type="xs:string" default="Austria"/>
  (of an "address" element)
  → `<address country=""/>`
  → `<address country="Hungary"/>`
  → `<address country="Austria"/>`
Defining a datatype

- Only simple types exist by default
  - Complex types must always be defined manually ⇒ see above!
  - Primitive types: Predefined in standard; very few
    - Example: string, date, time, datetime
  - Derived types: Derived from primitive types
    - Already derived in the standard
      - Examples: normalizedString, integer, unsignedByte
    - Or derived by the user
- They can be atomic, list or union
  - Atomic: Basic element; values can be restricted, enumerated, ...
  - List: Several atoms all of the same type; separated by whitespaces
  - Union: Combination of different types; can be any of those
Datatypes:

Built-in Datatype Hierarchy

anyType

all complex types

anySimpleType

duration

dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth

boolean

base64Binary

hexBinary

float
double

anyURI

QName

NOTATION

string

decimal

normalizedString

integer

token

nonPositiveInteger

long

nonNegativeInteger

language

Name

NMTOKEN

negativeInteger

int

unsignedLong

positiveInteger

NCName

NMTOKENS

short

unsignedInt

ID

IDREF

ENTITY

IDREFS

ENTITIES

byte

unsignedShort

unsignedByte

Image from the Schema specification: http://www.w3.org/TR/xmilschema-2/
Defining a datatype

Facets

- **Facet**: An independent dimension used for constraining the allowed values of a datatype
- **Fundamental facets**
  - "Properties" (fundamental qualities) of the datatype itself
    - What it is; important for primitive types
  - Cannot be changed
  - They are: Equal, ordered, bounded, cardinality, numeric
- **Constraining facets**
  - "Properties" of the values a datatype can contain
  - Define which values are allowed (optional)
  - Not all facets are available for all primitive types
    - E.g. no length, minLength, maxLength for date and time datatypes (date, dateTime, gMonth, etc.)
Defining a datatype
Constraining facets

- **length**: Characters, bytes, digits, list items: `<length value='8'/>`
- **minLength**: Minimum length: `<minLength value='1'/>`
- **maxLength**: Maximum length: `<maxLength value='30'/>`
- **pattern**: Regular expression: `<pattern value='[1-9][0-9]{3}'/>`
  - Very powerful; Perl-like regular expressions with some limitations
- **enumeration**: Set of allowed values
- **whiteSpace**: How to handle whitespaces (preserve, replace, collapse)
- **maxInclusive**: `<maxInclusive value='100'/>` (\(\leq 100\))
- **maxExclusive**: `<maxExclusive value='100'/>` (\(< 100\))
- **minInclusive**: `<minInclusive value='0'/>` (\(\geq 0\))
- **minExclusive**: `<minExclusive value='0'/>` (\(> 0\))
- **totalDigits**: `<totalDigits value='8'/>`
- **fractionDigits**: `<fractionDigits value='2'/>`
Datatype examples:

1. `<xs:simpleType name="clothingSize">`  
   `<xs:union>`  
   `<xs:simpleType>`  
   `<xs:restriction base="positiveInteger"/>`  
   `<xs:minInclusive value="40"/>`  
   `</xs:simpleType>`  
   `<xs:simpleType>`  
   `<xs:restriction base="string"/>`  
   `</xs:simpleType>`  
   `</xs:union>`  
   `</xs:simpleType>`

- "clothingSize" is either string or positive integer which is at least 40
- Valid content: "small", "XXL", 40, 54, "-1"
- Invalid content: 39, -1
Datatype examples:

- `<xs:simpleType name="orderIDType">`  
  `<xs:restriction base="xs:string">`  
  `<xs:pattern value="[0-9]{6}"/>`  
  `</xs:restriction>`  
  `</xs:simpleType>`

  - The base datatype of orderIDType is a string
  - It consists of exactly 6 ("{6}" ) numbers ("[0-9]"")
  - Valid content: "123456", "999999", "000000"
  - Invalid content: "12345", "12A456", 123456
    - The last is invalid because it is a number (e.g. integer) and not a string!

[Datatypes.xsd, Datatypes.xml]
There's even more!

- Empty elements, mixed content
- Include, redefine, import: Incorporating external content
- Identity constraint definitions
  - Uniqueness and reference constraints; similar to "ID" in DTDs
- Notations: Similar to "NOTATION" in DTDs
- Additional attributes
- Model groups
- Wildcards
- NIL values: Define / set something explicitly as missing
- Complex types: Deriving, restricting, redefining, abstract, controlling creation and use of derived types, ...
- Annotations
- ....
Schemas: Is it enough?

- Still missing: Semantical verification
  - Well-formedness and validity do not talk about the meaning
  - Date: Yes, it's correct; but what does it mean?
    » Is it the date the shipment must arrive, or the date of sending?
    » ...

- Unwieldy without an editor supporting schemas

- Steep learning phase for complicated parts
  - Difficult to master; didn't we hear this at the beginning?

- Sound design and adhering to design-rules necessary
  - Extending the restriction of the extension of the substitution ...

- Other competing type/version exist
  - E.g. RELAX NG by Oasis
  - Most applications however follow this specification!
Now we have defined our data according to both structure and content (datatypes)

But which enduser is ANYHOW interested in PURE XML???

Two versions for presentation possible:

- Parser: Applications use it as input and present it in their own way (graphics, UI, text, ...) to the user
  - Parsers exist for most programming languages
  - See part on programming with Java as example!
- Direct conversion to HTML: This is were XSL comes in!

XSL converts XML (back) to HTML
- Or something other, e.g. again to XML or any other text or even binary format
• Schema Specification: Primer (Introduction)
  http://www.w3.org/TR/xmlschema-0/
• Schema Specification: Structures
  http://www.w3.org/TR/xmlschema-1/
• Schema Specification: Datatypes
  http://www.w3.org/TR/xmlschema-2/
• RELAX NG
  http://www.oasis-open.org/committees/relax-ng/