Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 7 – XML Data Modeling

Outline

XML Origin and Usages

- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language, not a database language
  - Documents have tags giving extra information about sections of the document
  - For example:
    - <title> XML </title>
    - <slide> XML Origin and Usages </slide>
- Derived from SGML (Standard Generalized Markup Language)
  - standard for document description
    - enables document interchange in publishing, office, engineering, ...
  - main idea: separate form from structure
  - XML is simpler to use than SGML
    - roughly 20% complexity achieves 80% functionality
- XML (like SGML) is a meta-language
  - a language for the definition of languages (vocabularies)
  - examples
    - SGML -> HTML
    - XML -> XHTML

XML – Data and Metadata

- XML documents are to some extent self-describing
  - Tags (markup) represent metadata about specific parts/data items of a document
    - metadata provided at the 'instance'-level
  - Example

```
<bank>
    <account>
        <account-number> A-101     </account-number>
        <branch-name>    Downtown </branch-name>
        <balance>        500       </balance>
    </account>
    <depositor>
        <account-number> A-101     </account-number>
        <customer-name> Johnson </customer-name>
    </depositor>
</bank>
```

- Schema provides 'global' metadata (optional!)
  - defines the vocabulary, rules for document structure, permitted or default content
  - associated with/referenced by the document

Forces Driving XML

- Document Processing
  - Goal: use document in various, evolving systems
  - structure – content – layout
  - grammer: markup vocabulary for mixed content
- Data Bases and Data Exchange
  - Goal: data independence
  - structured, typed data – schema-driven – integrity constraints
- Semi-structured Data and Information Integration
  - Goal: integrate autonomous data sources
  - data source schema not known in detail – schemata are dynamic
  - schema might be revealed through analysis only after data processing

XML Documents

- XML documents are text (unicode)
  - markup (always starts with '<' or '&')
    - start/end tags
    - references (e.g., &lt, &amp, ...)
    - declarations, comments, processing instructions, ...
  - data (character data)
    - characters '<' and '&' need to be indicated using references (e.g., &lt) or using the character code
    - alternative syntax: <![CDATA[(a<b)&(c<d)]]>
- XML documents are **well-formed**
  - logical structure:
    [<declaration>] [<dtd>] [<comment-or-PI>] <element> [<comment-or-PI>]
    - (optional) XML declaration (XML version, encoding, ...)
    - (optional) schema (DTD)
    - single root element (possibly nested)
    - comments
    - processing instructions
      - example: reference to a stylesheet, used by a browser
  - additional requirements on the structure and content of <element>

XML Documents: Elements

- **Tag:** label for a section of data
- **Element**:
  - start tag <*tagname*>
  - content: text and/or nested element(s)
    - may be empty, alternative syntax: <*tagname/*>
  - end tag </*tagname*>
- Elements must be properly **nested** for the document to be **well-formed**
  - Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Mixture of text with sub-elements (mixed content) is legal in XML
  - Example:

```
<account>
   This account is seldom used any more.
    <account-number> A-102</account-number>
    <branch-name> Perryridge</branch-name>
    <balance>400 </balance>
</account>
```

  - Useful for document markup, but discouraged for data representation
- Element content (i.e., text and nested elements) is ordered!

XML Element Structure

- Arbitrary levels of nesting
- Same element tag can appear multiple times
  - at the same level

```
<bank-1>
  <customer>
      <customer-name> Hayes </customer-name>
      <account>
          <account-number> A-102 </account-number>
          <balance>          400 </balance>
      </account>
      <account> ... </account>
  </customer>
  ...
</bank-1>
```

  - at different levels

```
<product>
  <prodName>  ... </prodName>
  <part>
      <id> ... </id>
      <part> ... </part>
      <part> ... </part>
  </part>
  ...
</product>
```

XML Documents: Attributes

- **Attributes:** can be used to further describe elements
  - attributes are specified by *name="value"* pairs inside the starting tag of an element
  - value is a text string
    - no further structuring of attribute values
  - attributes are not ordered
- Example:

```
<account acct-type = "checking" >
        <account-number> A-102 </account-number>
        <branch-name> Perryridge </branch-name>
        <balance> 400 </balance>
</account>
```

- Well-formed documents:
  - attribute names must be unique within the element
  - attribute values are enclosed in single or double quotation marks

Attributes vs. Subelements

- Distinction between subelement and attribute
  - In the context of documents, attributes are part of markup, while subelement contents are part of the basic document content
    - markup used to interpret the content, influence layout for printing, etc.
  - In the context of data representation, the difference is unclear and may be confusing
    - Same information can be represented in two ways
      - `<account account-number = “A-101”> .... </account>`
      - `<account>`
        `<account-number>A-101</account-number> ...`
        `</account>`
- Limitations of attributes
  - single occurrence within element
  - no further attribute value structure, no ordering

Namespaces

- A single XML document may contain elements and attributes defined by different vocabularies
  - Motivated by modularization considerations, for example
- Name collisions have to be avoided
- Example:
  - A **Book** vocabulary contains a Title element for the title of a book
  - A **Person** vocabulary contains a Title element for an honorary title of a person
  - A **BookOrder** vocabulary uses both vocabularies
- Namespaces specifies how to construct universally unique names

Namespaces (cont.)

- Namespace is a collection of names identified by a URI
- Namespaces are declared via a set of special attributes
  - These attributes are prefixed by xmlns - Example:

```
<BookOrder xmlns:Customer="http://mySite.com/Person"
    xmlns:Item="http://yourSite.com/Book">
```

  - Namespace applies to the element where it is declared, and all elements within its content
    - unless overridden
- Elements/attributes from a particular namespace are prefixed by the name assigned to the namespace in the corresponding declaration of the using XML document
  - ...Customer:Title='Dr'...
  - ...Item:Title='Introduction to XML'...
- Default namespace declaration for fixing the namespace of unqualified names
  - Example:

```
<BookOrder xmlns="http://mySite.com/Person"
    xmlns:Item="http://yourSite.com/Book">
```

XML Document Schema

- XML documents may optionally have a schema
  - standardized data exchange, ...
- Schema restricts the structures and data types allowed in a document
  - document is **valid**, if it follows the restrictions defined by the schema
- Two important mechanisms for specifying an XML schema
  - Document Type Definition (DTD)
  - XML Schema

Document Type Definition - DTD

- Original mechanism to specify type and structure of an XML document
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- Special DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >
- DTD is
  - contained in the document, or
  - stored separately, referenced in the document
- DTD clause in XML document specifies the root element type, supplies or references the DTD
  - <!DOCTYPE bank [...]>

Element Specification in DTD

- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything defined in the DTD can be a subelement)
- Structure is defined using regular expressions
  - sequence (*subel*, *subel*, …), alternative (*subel* | *subel* | …)
  - number of occurences
    - "?" - 0 or 1 occurrence
    - "+" - 1 or more occurrences
    - "*" - 0 or more occurrences
- Example

```
<!ELEMENT depositor (customer-name,  account-number)>
<!ELEMENT customer-name(#PCDATA)>
<!ELEMENT account-number (#PCDATA)>
<!ELEMENT bank ( ( account | customer | depositor)+)>
```

Attribute Specification in DTD

- Attribute list of an element defines for each attribute
  - name
  - type of attribute (as relevant for data modeling)
    - character data (CDATA)
    - identifiers (ID) or references to an identifier attribute (IDREF, IDREFS)
      - see next chart for details
    - XML name tokens (NMTOKEN, NMTOKENS)
    - enumeration type
  - whether
    - mandatory (#REQUIRED)
    - default value (*value*)
    - optional without default (#IMPLIED), or
    - the value, if present, must not differ from the given one (#FIXED *value*)
- Examples
  - <!ATTLIST account acct-type CDATA "checking">
  - <!ATTLIST customer

```
customer-id   ID       #REQUIRED
accounts      IDREFS   #REQUIRED   >
```

IDs and IDREFs

- An element can have at most one attribute of type ID
- The ID attribute value of each element in an XML document must be distinct

  ➔ ID attribute (value) is an object identifier
- An attribute of type IDREF must contain the ID value of an element in the same document
- An attribute of type IDREFS contains a set of (0 or more) ID values. Each ID value must contain the ID value of an element in the same document
- IDs and IDREFs are untyped, unfortunately
  - Example below: The *owners* attribute of an account may contain a reference to another account, which is meaningless;
    owners attribute should ideally be constrained to refer to customer elements

Example: Extended Bank DTD

- Bank DTD with ID and IDREF attribute types

```
<!DOCTYPE bank [
    <!ELEMENT account (branch-name, balance)>
    <!ATTLIST account
            account-number   ID       #REQUIRED
            owners           IDREFS   #REQUIRED>
     <!ELEMENT customer(customer-name, customer-street,
                                                customer-city)>
     <!ATTLIST customer
            customer-id      ID              #REQUIRED
            accounts         IDREFS      #REQUIRED>

     ... declarations for bank, branch-name, balance, customer-name,
                        customer-street and customer-city

]>
```

XML data with ID and IDREF attributes

```
<bank>
    <account account-number="A-401" owners="C100 C102">
        <branch-name> Downtown </branch-name>
        <balance>500 </balance>
    </account>
        . . .
    <customer customer-id="C100" accounts="A-401">
        <customer-name>Joe</customer-name>
        <customer-street>Monroe</customer-street>
        <customer-city>Madison</customer-city>
    </customer>
    <customer customer-id="C102" accounts="A-401 A-402">
        <customer-name> Mary</customer-name>
        <customer-street> Erin</customer-street>
        <customer-city> Newark </customer-city>
    </customer>
</bank>
```

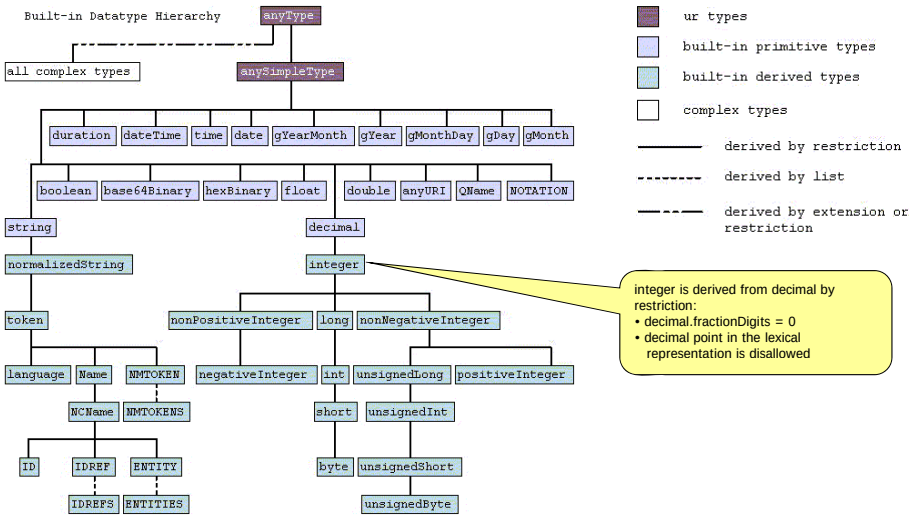

Schema Definition with XML Schema

- XML Schema is closer to the general understanding of a (database) schema
- XML Schema (unlike DTD) supports
  - Typing of values
    - E.g. integer, string, etc
  - Constraints on min/max values
  - Typed references
  - User defined types
  - Schema specification in XML syntax
    - schema is a well-formed and valid XML document
  - Integration with namespaces
  - Many more features
    - List types, uniqueness and foreign key constraints, inheritance ..
- BUT: significantly more complicated than DTDs

Types in XML Schema

- Simple vs. complex types
  - Simple type
    - no further structure, does not contain child elements or attributes
    - can be used as a type for both attribute values and element content
    - broad repertoire of pre-defined simple types
    - facets of simple types provide additional characteristics
      - e.g., pattern, length
  - Complex type
    - consists of attribute declarations (optional) and a content model
    - content model defines possible child elements, content based on simple types, mixed content
- Primitive vs. derived types
  - Primitive types
    - subset of the simple types that are not defined in terms of other types
      - Examples: string, decimal
  - Derived types
    - defined in terms of other (derived or primitive) base types
    - different derivation mechanisms
      - by restriction – derived type permits only subset of value or literal space of the base type
      - by list, union – similar to composite types
      - by extension – similar to subtyping
- Built-in vs. user-derived types

XML Schema Built-in Types



Derivation By Restriction

- Based on the following facets
  - upper/lower bounds for value domain
    - minExclusive, minInclusive
    - maxExclusive, maxInclusive
  - length for strings, names, URIs or lists
    - length
    - maxLength
    - minLength
  - length restrictions for decimal
    - totalDigits
    - fractionDigits
  - value enumeration
    - enumeration
  - regular expression limiting the lexical space
    - pattern
- Examples
  - <xs:simpleType name="MoneyAmnt">
    <xs:restriction base="xs:decimal">
    <xs:totalDigits value="10"/>
    <xs:fractionDigits value="2"/>
    </xs:restriction>
    </xs:simpleType>
  - <xs:simpleType name="Phone"
    <xs:restriction base="xs:string">
    <xs:pattern
    value="0[1-9][0-9]+\-[1-9][0-9]+"/>
    </xs:restriction>
    </xs:simpleType>

Complex Types

- Needed for modeling attributes and content model of elements
  - defines the type of the element, but not the element tag name
- Simple content: no child elements, extends/restricts a simple type for element content
  - <xs:complexType name="Money">
    <xs:simpleContent>
    <xs:extension base="MoneyAmt">
    <xs:attribute name="currency" type="xs:string" use="required"/>
    </xs:extension>
    </xs:simpleContent>
    </xs:complexType>

Complex Types (cont.)

- Complex content
  - three types of content models (may be nested arbitrarily)
    - sequence – subelements have to occur in the specified order
    - choice – only one of the subelements may occur
    - all – each subelement can appear at most once, in arbitrary order

```
<xs:complexType name="AccountT">
    <xs:sequence>
        <xs:element name="account-number" type="xs:string"/>
        <xs:element name="branch-name"    type="xs:string"/>
        <xs:element name="balance"        type="Money"/>
    </xsd:sequence>
</xs:complexType>
```

- Specifying the number of occurences
  - minOccurs, maxOccurs attributes can be used in element and content model definitions
    - <xs:element name="account" type="AccountT minOccurs="0" maxOccurs="10"/>
    - <xs:choice minOccurs="2" maxOccurs="unbounded"> ... </xs:choice>

Restricting And Extending Complex Types

- Derivation by restriction
  - derived type has the same content model as the base type in terms of valid attributes, elements
  - restrictions possible by
    - limiting the number of occurrences by chosing a larger min or smaller max value
    - supplying a default or fixed attribute value
    - remove an optional component
    - replacing a simple type with a derivation of the simple type
- Derivation by extension
  - new attributes and elements can be added to the type definition inherited from the base type
    - append-only for elements, implying a sequence model

```
<xs:complexType name="SavingsAccountT">
  <xs:complexContent>
    <xs:extension base="AccountT">
      <xs:sequence>
          <xs:element name="interest-rate" type="xs:decimal"/>
      </xsd:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Derived Types and "Substitutability"

- Derived types can be explicitly used in schema definitions
- At the document (i.e., "instance") level
  - an instance of a derived type may appear instead of an instance of its base type
    - derivation by extension or by restriction
    - may be explicitly blocked for a base type in the schema definition
  - the derived type has to be indicated using xsi:type
    - example (assuming that element account has type AccountT):

```
<account xsi:type="SavingsAccountT">
        <account-number>1234</account-number>
        <branch-name>Kaiserslautern</branch-name>
        <balance currency="Euro">3245.78</balance>
        <interest-rate>3.5</interest-rate>
</account>
```

  - the element name is not affected, only the content
- Substitution groups
  - extends the concept to the element level
  - a named head element may be substituted by any element in the substitution group
    - group elements have to be derived from head element
- Elements and types may be declared as "abstract"

Namespaces and XML Schema

- XML-Schema elements and data types are imported from the XML-Schema namespace http://www.w3.org/2001/XMLSchema
  - **xsd** is generally used as a prefix
- The vocabulary defined in an XML Schema file belongs to a target namespace
  - declared using the **targetNamespace** attribute
  - declaring a target namespace is optional
    - if none is provided, the vocabulary does not belong to a namespace
    - required for creating XML schemas for validating (pre-namespace) XML1.0 documents
- XML document using an XML schema
  - declares namespace, refers to the target namespace of the underlying schema
  - can provide additional hints where an XML schema (xsd) file for the namespace is located
    - schemaLocation attribute

XML Schema Version of Bank DTD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.banks.org"
            xmlns ="http://www.banks.org" >
<xsd:element name="bank" type="BankType"/>
<xsd:element name="account">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="account-number" type="xsd:string"/>
            <xsd:element name="branch-name"    type="xsd:string"/>
            <xsd:element name="balance"        type="xsd:decimal"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>                  ..... definitions of customer and depositor ....
<xsd:complexType name="BankType">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element ref="account"/>
        <xsd:element ref="customer"/>
        <xsd:element ref="depositor"/>
    </xsd:choice>
</xsd:complexType>
</xsd:schema>
```


XML Document Using Bank Schema

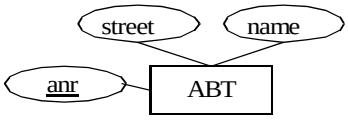
```
<bank   xmlns="http://www.banks.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.banks.org  Bank.xsd">
    <account>
        <account-number> ... </account-number>
        <branch-name> ... </branch-name>
        <balance> ... </balance>
    </account>
    ...
</bank>
```

Assertions in XML-Schema

- Uniqueness: UNIQUE-Element, KEY-Element
  - forces uniqueness of attribute or element values
    - <field> element(s)
  - can be applied to/declared for specific parts of the XML document
    - <selector> element
  - Example: within a bank element, all accounts should have a unique account number
    - <xs:element name="bank" type="bankType">
      <xs:unique name="uniqueAcctNo">
      <xs:selector xpath="/account"/>
      <xs:field xpath="account-number"/>
      </xs:unique>
      </xs:element>
- Some remarks
  - NULL value semantics: **nillable** at the schema level, **nil** in the document
  - *<key>* equivalent to *<unique> and nillable="false"*
  - composite keys/unique elements

Mapping ER-Model -> XML Schema

- Mapping Entities
  - 1:1 mapping to XML elements
  - use <key> to represent ER key attributes




```
...
  <element name="ABT">
    <complexType>
          <attribute name="anr" type="string" />
          <attribute name="street" type="string" />
          <attribute name="name" type="string" />
    </complexType>
  </element>
...
  <key name="abt_pk">
    <selector xpath="./ABT" />
    <field xpath="@anr" />
  </key>
...
```

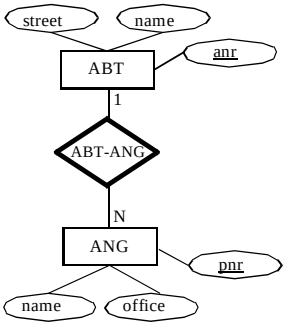
Mapping 1:N Relationships

- Mapping alternative: nesting
  - using local element definition

```
<element name="ABT">
    <complexType>
      <sequence>
        <element name="ANG">
          <complexType>
            <attribute name="name" type="string"/>
            <attribute name="office" type="string"/>
          </complexType>
        </element>
      </sequence>
      <attribute name="street" type="string"/>
      <attribute name="name" type="string"/>
    </complexType>
</element>
```

  - using global element definition

```
<element name="ABT">
    <complexType>
      <sequence>
        <element ref="ANG">
      </sequence>
      <attribute name="street" type="string" />
      <attribute name="name" type="string" />
    </complexType>
</element>
```




```
...
<element name="ANG">
    <complexType>
      <attribute name="name" type="string" />
      <attribute name="office" type="string" />
    </complexType>
</element>
```


Primary/Foreign Keys

- Problem
  - nesting alone is not sufficient for modeling a 1:n relationship
  - element identity is required to avoid duplicate entries
- Foreign Keys
  - guarantee referential integrity: <key> / <keyref> elements

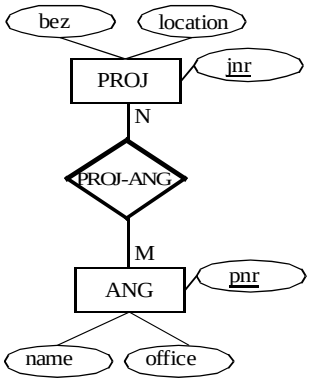
```
<element name="ABT">
    <complexType>
      <sequence>
        <element name="ANG">
          <complexType>
            <attribute name="pnr" type="string"/>
            <attribute name="name" type="string"/>
            <attribute name="office" type="string"/>
            <attribute name="abtid" type="string"/>
          </complexType>
        </element>
      </sequence>
      <attribute name="anr" type="string"/>
      <attribute name="name" type="string"/>
      <attribute name="street" type="string"/>
    </complexType>
</element>
...
```

```
...
<key name="abt_pk">
        <selector xpath=".//ABT" />
        <field xpath="@anr" />
</key>
...
<key name="ang_uniq">
        <selector xpath=".//ABT/ANG" />
        <field xpath="@pnr" />
</unique>
...
<keyref name="abt_fk" refer="abt_pk">
        <selector xpath=".//ABT/ANG" />
        <field xpath="@abtid" />
</key>
...
```

Primary/Foreign Keys

- Advantages over ID/IDREF
  - based on equality of data types
  - composite keys
  - locality, restricting scope to parts of the XML document
- Mapping of N:M – relationships
  - use <key>/<keyref> elements
  - flat modeling plus "pointers"
  - addition of helper element similar to mapping to relational model

```
<element name="PROJ_ANG">
    <complexType>
      <attribute name="pnr" type="string" />
      <attribute name="jnr" type="string" />
    </complexType>
</element>
```



Summary

- XML introduction and overview
  - document structure – elements, attributes
  - namespaces
- XML schema support
  - document type definitions (DTD)
    - document structure, but no support for data types, namespaces
  - XML Schema specification
    - powerful: structure, data types, complex types, type refinement, constraints, ...
    - complex!
- Mapping ER -> XML
  - 1:1, 1:n, n:m relationships
  - primary/foreign keys