AG Heterogene Informationssysteme
Prof. Dr.-Ing. Stefan Deßloch
Fachbereich Informatik
Technische Universität Kaiserslautern

Recent Developments for Data Models – Solution to Exercise 5

Monday, July 2, 2012 – 15:30 to 17:00 – Room 36-336

1) XML Data Modelling

XML Schema allows defining and describing a class of XML documents by using schema components to constrain and document the meaning, usage and relationships of their constituent parts: data types, elements and their content, and attributes and their values.

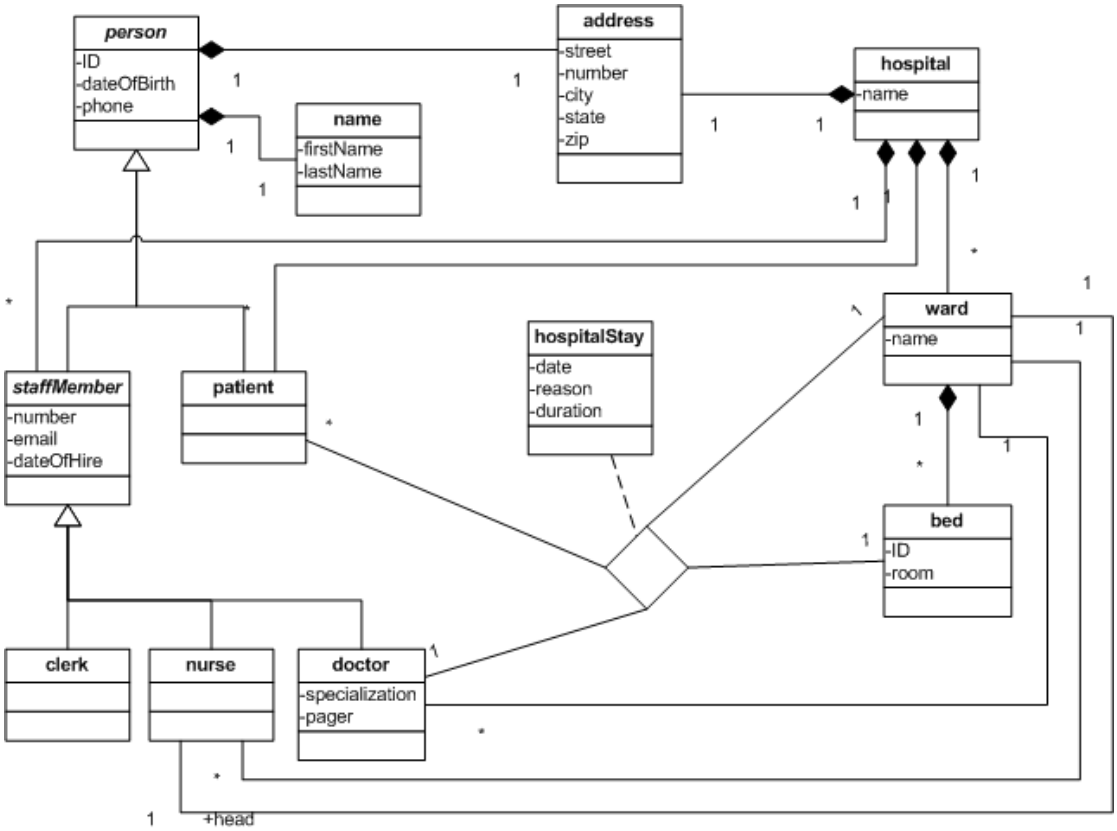

Figure 1: Hospital scenario

Consider the following scenario[1]: Hospitals have a name and an address. A hospital consists of several wards with a unique name and a number of beds. Each ward is

[1] This exercise is based on *Wolfgang Lehner, Harald Schöning: XQuery – Grundlagen und fortgeschrittene Methoden, dpunkt.verlag, 2004*

headed by a nurse. The hospital is used by different kinds of persons. First, there are patients. Second, there are staff members that can be distinguished into doctors, nurses, and clerks. Doctors are specialists in surgery, anesthesia, oncology, otology, or orthopedics. Doctors are provided with pagers. Each person has a unique identifier, a name consisting of one or more first names and a single last name, a date of birth, and a phone number. For staff members the employee number, the email address, and the date of hire are also relevant. Doctors and nurses work in one of the hospital's wards. When patients stay in the hospital, they are assigned to a ward and provided with a bed. For the time of the stay one of the doctors is responsible for the patient. The start date, the duration, and the reason for the stay are documented. An overview of the scenario is provided by Figure 1.

Specify an XML Schema file based on the hospital scenario!

a. Declare restricted derived types for international phone numbers and email addresses.

```
<xs:simpleType name="phone_T">
  <xs:restriction base="xs:string">
    <xs:pattern value="((([+]|00)[1-9]{1,2} )|0)[1-9][0-9-]+" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="email_T">
  <xs:restriction base="xs:string">
    <xs:pattern
      value="[A-Za-z.\-_]+@{1}[A-Za-z.\-_]+[.]{1}[A-Za-z]{2,3}" />
  </xs:restriction>
</xs:simpleType>
```

b. Declare a complex type for a postal address.

```
<xs:complexType name="address_T">
  <xs:sequence>
    <xs:element name="street" type="xs:string" />
    <xs:element name="number" type="xs:string" />
    <xs:element name="city" type="xs:string" />
    <xs:element name="state" type="xs:string" />
    <xs:element name="zip">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="[0-9]{5}" />
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

c. Declare a type hierarchy for persons, staff members, clerks, nurses, and doctors. Note that the types for persons and staff members will not be instantiated directly.

```
<xs:complexType name="person_T" abstract="true">
  <xs:sequence>
    <xs:element name="name">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="firstName" type="xs:string" minOccurs="1"
maxOccurs="unbounded" />
          <xs:element name="lastName" type="xs:string" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="address" type="address_T" />
    <xs:element name="dateOfBirth" type="xs:date" />
    <xs:element name="phone" type="phone_T" />
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required" />
</xs:complexType>

<xs:complexType name="staffMember_T" abstract="true">
  <xs:complexContent>
    <xs:extension base="person_T">
      <xs:sequence>
        <xs:element name="number" type="xs:string" />
        <xs:element name="email" type="email_T" minOccurs="0" />
        <xs:element name="dateOfHire" type="xs:date" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="doctor_T">
  <xs:complexContent>
    <xs:extension base="staffMember_T">
      <xs:sequence>
        <xs:element name="specialization">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="surgery" />
              <xs:enumeration value="anesthesia" />
              <xs:enumeration value="oncology" />
              <xs:enumeration value="otology" />
              <xs:enumeration value="orthopedy" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="pager" type="xs:string" />
      </xs:sequence>
      <xs:attribute name="ward" type="xs:string" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="clerk_T">
  <xs:complexContent>
    <xs:extension base="staffMember_T" />
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="nurse_T">
  <xs:complexContent>
    <xs:extension base="staffMember_T">
      <xs:attribute name="ward" type="xs:string" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

d. Declare elements for the staff member type hierarchy. Note that the staff member element should be substitutable by its sub-elements.

```
<xs:element name="staffMember" type="staffMember_T"
    abstract="true" />
<xs:element name="doctor" type="doctor_T"
    substitutionGroup="staffMember" />
<xs:element name="clerk" type="clerk_T"
    substitutionGroup="staffMember" />
<xs:element name="nurse" type="nurse_T"
    substitutionGroup="staffMember" />
```

e. Declare a derived type for patients and elements for the remaining entities. What alternatives do you see to model relationships between different entities?

There are three main techniques to express relationships in the XML data model. The first alternative is using the containment hierarchy. A complex type definition may be used to nest dependent entities and express 1-to-n relationships.

In the scenario it seems natural to nest the `hospitalStay` elements under the associated `patient` element.

As a second alternative attributes of type ID can be used to assign elements with a unique identifier. Such an element may be referenced using attributes of the type IDREF or IDREFS. The value of an IDREF attribute must match the value of some ID attribute in the document. The value of IDREFS attribute can contain several references to elements with ID attribute separated with whitespaces.

However, IDs and IDREFs have some weaknesses: IDs must be unique across an entire document, and IDREF declarations do not specify the type of element an instance of the IDREF attribute must reference. XML Schema provides a way to specify relationships in much the same way that foreign-key relationships are declared in a relational database. As opposed to IDs, keys may be strongly typed in XML Schema. Furthermore, the definition of composite keys is possible.

In the following, the key/keyref mechanism is used to express the relationships between doctors and wards, nurses and wards, head nurses and wards, and patients staying in the hospital, doctors, wards, and beds.

```
<xs:element name="ward">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="bed" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="ID" type="xs:ID" use="required" />
          <xs:attribute name="room" type="xs:string"
              use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="head" type="xs:IDREF" use="required" />
  </xs:complexType>
</xs:element>

<xs:complexType name="patient_T">
  <xs:complexContent>
    <xs:extension base="person_T">
      <xs:sequence>
        <xs:element ref="hospitalStay" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="hospital">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" />
      <xs:element name="address" type="address_T" />
      <xs:element name="wards">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="ward" maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="staff">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="staffMember" maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="patients">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="patient" type="patient_T"
                maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:key name="doctor_pk">
    <xs:selector xpath="./staff/doctor" />
    <xs:field xpath="@ID" />
  </xs:key>
```

```
  <xs:keyref name="doctor_ward_fk" refer="ward_pk">
    <xs:selector xpath="./staff/doctor" />
    <xs:field xpath="@ward" />
  </xs:keyref>
  <xs:key name="ward_pk">
    <xs:selector xpath="./wards/ward" />
    <xs:field xpath="name" />
  </xs:key>
  <xs:keyref name="ward_nurse_fk" refer="nurse_pk">
    <xs:selector xpath="./wards/ward" />
    <xs:field xpath="@head" />
  </xs:keyref>
  <xs:key name="nurse_pk">
    <xs:selector xpath="./staff/nurse" />
    <xs:field xpath="@ID" />
  </xs:key>
  <xs:keyref name="nurse_ward_fk" refer="ward_pk">
    <xs:selector xpath="./staff/nurse" />
    <xs:field xpath="@ward" />
  </xs:keyref>
  <xs:key name="clerk_pk">
    <xs:selector xpath="./staff/clerk" />
    <xs:field xpath="@ID" />
  </xs:key>
  <xs:key name="bed_pk">
    <xs:selector xpath="./wards/ward/bed" />
    <xs:field xpath="@ID" />
  </xs:key>
</xs:element>


<xs:element name="hospitalStay">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="date" type="xs:date" />
      <xs:element name="duration" type="xs:integer" />
      <xs:element name="reason" type="xs:string" />
      <xs:element name="doctor" type="xs:string" />
      <xs:element name="ward" type="xs:string" />
      <xs:element name="bed" type="xs:string" />
    </xs:sequence>
  </xs:complexType>

  <xs:keyref name="doctor_hospitalStay_fk" refer="doctor_pk">
    <xs:selector xpath="." />
    <xs:field xpath="doctor" />
  </xs:keyref>
  <xs:keyref name="ward_hospitalStay_fk" refer="ward_pk">
    <xs:selector xpath="." />
    <xs:field xpath="ward" />
  </xs:keyref>
  <xs:keyref name="bed_hospitalStay_fk" refer="bed_pk">
    <xs:selector xpath="." />
    <xs:field xpath="bed" />
  </xs:keyref>
</xs:element>
```

2) XQuery

Specify XQuery expressions based on the XML schema defined in 1) to produce the following results.

a. All doctor elements enclosed in a root element called `listOfDoctors`.

```
<listOfDoctors>
{
for $a in fn:doc("doc.xml")//staff/doctor
return $a
}
</listOfDoctors>
```

b. The first and last name of all doctors living in “Berlin” enclosed in a root element called `doctorsFromBerlin` in alphabetical order.

```
<doctorsFromBerlin>
{
for $a in fn:doc("doc.xml")//staff/doctor
let $firstName := $a//firstName,
    $lastName := $a//lastName
where $a//city = 'Berlin'
order by $lastName, $firstName
return
    <doctor> {$lastName, $firstName} </doctor>
}
</doctorsFromBerlin>
```

c. The name of each ward together with the last and first name of its head nurse.

```
let $a := fn:doc("doc.xml")
for $w in $a//ward, $n in $a//nurse
where $w/@head = $n/@ID
return <ward>
         {$w/name}
         <head> {$n//lastName, $n//firstName} </head>
       </ward>


for $w in fn:doc("doc.xml")//ward
let $n := fn:id($w/@head)
return
    <ward>
    {$w/name}
        <head>
        {$n//lastName, $n//firstName}
        </head>
    </ward>
```

d. The name of each ward together with the names of all associated nurses.

```
let $a := fn:doc("doc.xml")
for $w in $a/hospital/wards/ward
return
    <ward>
        {$w/name}
        <nurses>
        {
        for $n in $a//nurse
        where $n/@ward = $w/name/text()
        return
            $n//name
        }
        </nurses>
    </ward>
```

e. The name of each ward together with the *number* of associated nurses.

```
let $a := fn:doc("doc.xml")
for $w in $a/hospital/wards/ward
return
    <ward>
    {$w/name}
        <nurses>
        {
        fn:count(for $n in $a//nurse
                 where $n/@ward = $w/name/text()
                 return $n)
        }
        </nurses>
    </ward>
```

f. The name of each ward together with a sequence number indicating its position in the XML document.

```
for $w at $i in fn:doc("doc.xml")/hospital/wards/ward
return
    element ward {
    attribute number {$i},
    $w/name/text()}
```

g. The names of all employees ordered by the data of hire starting with the most recent one. Missing values should appear at the end of the list.

```
for $d in fn:doc("doc.xml")//dateOfHire
order by $d descending empty greatest
return $d/parent::*/name
```

h. The city names in alphabetical order with the names of all residents.

```
let $a := fn:doc("doc.xml")
for $c in fn:distinct-values($a//city)
order by $c
return
  <city>
    <name>{$c}</name>
    {
    for $p in $a//(doctor | nurse | clerk | patient)
    (: or $a//element(*, Person_T) alternatively :)
    where $p//city = $c
    return $p/name
    }
  </city>
```

i. All patients together with a list of doctors living in the same city.

```
let $a := fn:doc("doc.xml")
for $p in $a//patient
let $d := $a//doctor[.//city = $p//city]
return
    <patient>
        {$p//name}
        <doctors>
            { $d//name }
        </doctors>
    </patient>
```

j. The average length each patient stood in the hospital.

```
for $p in fn:doc("doc.xml")//patient
let $s := $p/hospitalStay/duration
return
    <patient>
        {$p//name}
        <averageDuration>
            { fn:avg($s) }
        </averageDuration>
    </patient>
```

k. The number of doctors that are specialist for "surgery", "anesthesia", "oncology".

```
for $s in ("surgery", "anesthesia", "oncology")
let $d :=
    fn:doc("doc.xml")//doctor[./specialization = $s]
return
    <specialization>
        {$s}
        <doctors>
            { fn:count($d) }
        </doctors>
    </specialization>
```

l. A numbered list of staff members ordered by date of birth. Staff members with an unspecified date of birth shall appear at the end of the list.

```
for $s2 at $i in (
  for $s in
    fn:doc("doc.xml")//staff//(doctor | nurse | clerk)
    (: or element(*, staffMember_T) alternatively :)
  order by $s//dateOfBirth ascending empty greatest
  return $s)
return element staffMember {
  element number {$i}, $s2//name
}
```

m. A list of specializations. Doctors shall be nested within their respective specialization.

```
for $s in
  fn:distinct-values(fn:doc("doc.xml")//specialization)
return element specialization {
  attribute name {$s},
  for $d in fn:doc("doc.xml")//staff//doctor
  where $d//specialization = $s
  return $d/name
}
```

n. Retrieve the date of birth of the youngest and the oldest staff member for each group (clerks, doctors, and nurses).

```
let $doc := fn:doc("doc.xml")
for $a in fn:distinct-values(
    for $s in $doc//staff/*
    return fn:node-name($s)
    )
let $g := $doc//staff/*[fn:node-name(.) = $a]
return element {$a} {
    element minDateOfBirth {
        fn:min($g//dateOfBirth)
    },
    element maxDateOfBirth {
        fn:max($g//dateOfBirth)
    }
}
```

o. All patients that have been treated at the emergency ward.

```
for $p in fn:doc("doc.xml")//patient
where some $w in $p/hospitalStay/ward
    satisfies ($w = "emergency ward")
return $p/name

(: the following query is equivalent :)
for $p in fn:doc("doc.xml")//patient
where $p/hospitalStay/ward = "emergency ward"
return $p/name

(: however, the following query produces an error,
because a sequence of more than one item is not allowed
as the first operand of 'eq' :)
for $p in fn:doc("doc.xml")//patient
where $p/hospitalStay/ward eq "emergency ward"
return $p/name
```

p. All patients that have been treated at the emergency ward only and have not been to any other ward.

```
for $p in fn:doc("doc.xml")//patient
where every $w in $p/hospitalStay/ward
    satisfies ($w = "emergency ward")
return $p/name
```

q. Patients that have been treated in two or more wards.

```
for $p in fn:doc("doc.xml")//patient
let $w := fn:distinct-values($p/hospitalStay/ward)
where fn:count($w) >= 2
return $p/name
```