From Anchor Model to XML

L. Rönnbäck
lars.ronnback@resight.se
Resight, Kungsholms Strand 179, 112 48 Stockholm

O. Regardt
olle.regardt@teracom.se
Teracom, Kaknästornet, 102 52 Stockholm

M. Bergholtz, P. Johannesson, P. Wohed
maria@dsv.su.se, pajo@dsv.su.se, petia@dsv.su.se
DSV, Stockholm University, Sweden

September 15, 2010

An anchor schema can be presented in XML. In this paper we define an XML anchor schema definition and discuss major points with the solution. We use the anchor model in Figure 1 as an example and show how it can be presented in XML.

The listing in Figure 2 shows the XML Schema definition of the anchor schema and the listing in Figure 3 shows the implemented XML schema for the running example in Figure 1.

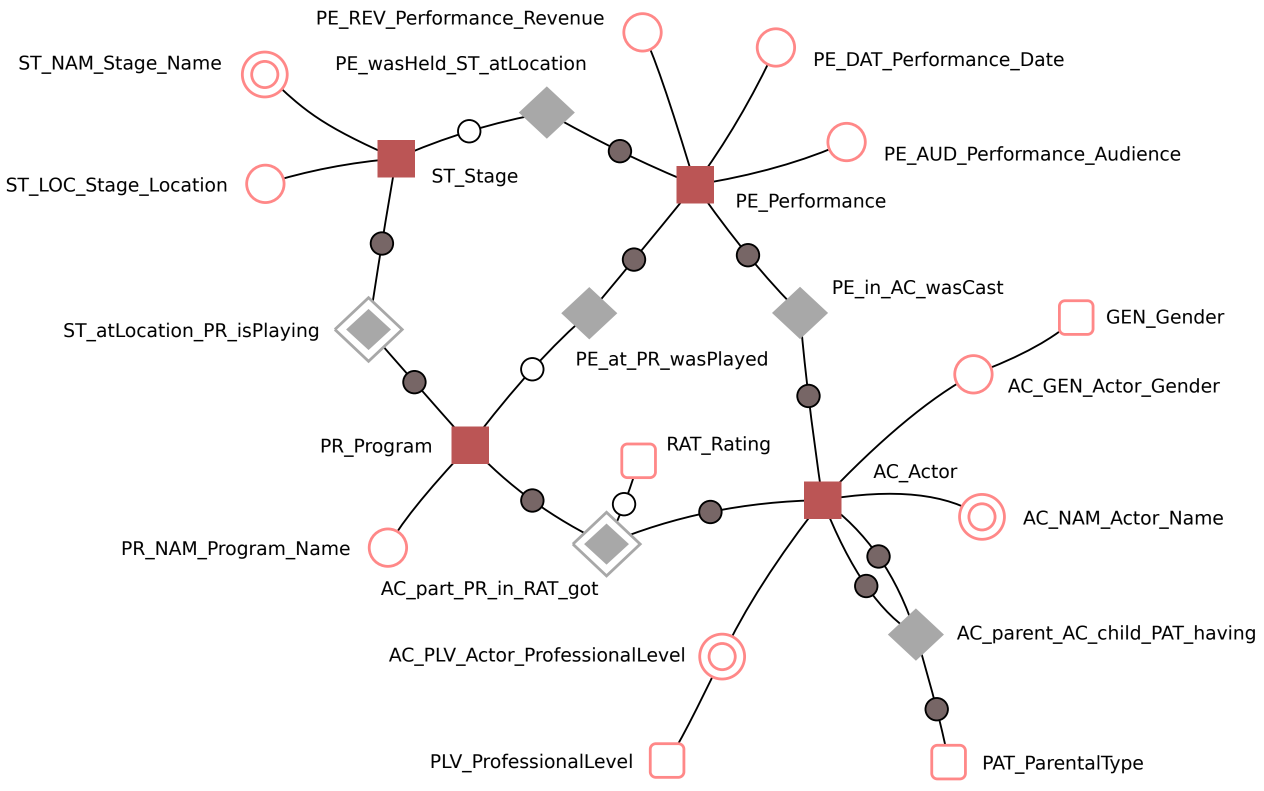

Figure 1: An anchor model

In the XML Schema definition (lines 6–14 in Figure 2), three simple types are introduced, identity, dataType and timeType to capture the basic notions in anchor modeling i.e., identities, data type and time type. In addition the simple type name is introduced (lines 3–5) to unify the presentation of names present for most of the elements of an anchor schema. The building stones of an anchor schema, i.e. anchor, knot, attribute and tie are captured through corresponding complex types in the XML Schema definition (lines 20–49). The four variations of attributes, i.e. static, historized, knotted static, knotted historized, are captured through a combination of the (optional) attributes knotRange, dataRange and timeRange (lines 26–33). Static attributes have only a dataRange (e.g. see attributes on lines 20–22 in Figure 3). Historized attributes have a dataRange and a timeRange (see the attribute 'Name' on line 16), knotted static attributes have a knotRange (see the 'Gender' attribute on line 9), and knotted historized attributes have a knotRange and a timeRange (see the attribute 'ProfessionaLevel' on line 10).

In contrast to the structure of the formal definitions where an attribute refers to an anchor (i.e. an attribute has an anchor as a domain), in the XML Schema definition the attributes are captured as elements in the hierarchical structure of the anchors being their domains (lines 35–37 in Figure 2). The anchor, attribute and knot complex types all contain mnemonic and descriptor attributes (see the naming conventions presented in ???). The mnemonic is a short string used to uniquely identify an element and in the case of knots and anchors, also as a referent to which attributes and ties can refer. Descriptors are longer strings meant as verbal descriptions of the objects being modeled. The anchor and knot complex types contain, in addition, an identity attribute (corresponding to their domains).

In accordance with to the structure of the formal definition, ties are compound of at least two anchorRole elements and a number of knotRole elements. The four variations of ties are captured through a combination of the contained elements and attributes. A static tie has only anchorRole elements (e.g. see lines 29–32 in Figure 3), a historized tie has anchorRole elements and a timeRange attribute (see lines 46–49), a knotted static tie has anchorRole and knotRole elements (see lines 24–28), and finally a knotted historized tie has anchorRole and knotRole elements as well as a timeRange attribute (see lines 41–45). At least one anchorRole in the tie must also have its identifier attribute set to true.

Finally, an anchor schema contains a number of knots, anchors (containing a number of attributes) and ties (lines 52–54 in Figure 2). To capture the conditions that an anchor schema must satisfy, a number of assertions[1] are defined (lines 56–63). These are XPath expressions which should be true for any given implementation of the XML Schema. For example the assertion "every $k in anchor/attribute/@knotRange satisfies knot[@mnemonic = $k]" (line 56) states that, for every attribute (found in anchors) that has a knotRange there must exist a knot in the schema whose mnemonic is equal to that knotRange. This guarantees referential integrity between knotted attributes and knots in the schema. Another type of assertion is "not(some $t1 in tie, $t2 in $t1/preceding-sibling::tie satisfies deep-equal($t1,$t2))" (line 59) stating that for no tie an exact copy of that tie may exist. This guarantees the uniqueness of ties in the schema.

The XML representation can be used as an interchange format for anchor schemas. It is also suitable for automatic code generation. Using XSLT the schema in Figure 3 can be transformed into SQL code. When a schema is extended, for example with some new attributes, new code can be generated and run to non-destructively reflect those changes in the database. Furthermore, the schema can also be versioned using a revision control system in order to keep track of such schema changes.

[1]Assertions are supported in XML Schema version 1.1 and expressed using XPath version 2.0 syntax.

```
<?xml version=”1.0”?>
<xs:schema
  xmlns:xs=”http://www.w3.org/2001/XMLSchema”
  xmlns=”http://anchormodeling.com/schema”
  targetNamespace=”http://anchormodeling.com/schema”
  xpathDefaultNamespace=”##targetNamespace”
  elementFormDefault=”qualified”>
   <xs:simpleType name=”name”>
      <xs:restriction base=”xs:string”/>
   </xs:simpleType>
   <xs:simpleType name=”identity”>
      <xs:restriction base=”xs:string”/>
   </xs:simpleType>
   <xs:simpleType name=”dataType”>
      <xs:restriction base=”xs:string”/>
   </xs:simpleType>
   <xs:simpleType name=”timeType”>
      <xs:restriction base=”xs:string”/>
   </xs:simpleType>
   <xs:complexType name=”role”>
      <xs:attribute name=”role” type=”name”
        use=”required”/>
      <xs:attribute name=”type” type=”name”
        use=”required”/>
      <xs:attribute name=”identifier” type=”xs:boolean”
        use=”optional” default=”false”/>
   </xs:complexType>
   <xs:complexType name=”knot”>
      <xs:attribute name=”mnemonic” type=”name”
        use=”required”/>
      <xs:attribute name=”descriptor” type=”name”
        use=”required”/>
      <xs:attribute name=”identity” type=”identity”
        use=”required”/>
      <xs:attribute name=”dataRange” type=”dataType”
        use=”required”/>
   </xs:complexType>
   <xs:complexType name=”attribute”>
      <xs:attribute name=”mnemonic” type=”name”
        use=”required”/>
      <xs:attribute name=”descriptor” type=”name”
        use=”required”/>
      <xs:attribute name=”knotRange” type=”name”
        use=”optional”/>
      <xs:attribute name=”dataRange” type=”dataType”
        use=”optional”/>
      <xs:attribute name=”timeRange” type=”timeType”
        use=”optional”/>
      <xs:assert test=”count(@knotRange) +
        count(@dataRange) = 1”/>
   </xs:complexType>
   <xs:complexType name=”anchor”>
      <xs:sequence>
         <xs:element name=”attribute” type=”attribute”
           minOccurs=”1” maxOccurs=”unbounded”/>
      </xs:sequence>
      <xs:attribute name=”mnemonic” type=”name”
        use=”required”/>
      <xs:attribute name=”descriptor” type=”name”
        use=”required”/>
      <xs:attribute name=”identity” type=”identity”
        use=”required”/>
   </xs:complexType>
   <xs:complexType name=”tie”>
      <xs:sequence>
         <xs:element name=”anchorRole” type=”role”
           minOccurs=”2” maxOccurs=”unbounded”/>
         <xs:element name=”knotRole” type=”role”
           minOccurs=”0” maxOccurs=”unbounded”/>
      </xs:sequence>
      <xs:attribute name=”timeRange” type=”timeType”
        use=”optional”/>
      <xs:assert test=”anchorRole[@identifier = ’true’]”/>
   </xs:complexType>
   <xs:complexType name=”schema”>
      <xs:choice minOccurs=”0” maxOccurs=”unbounded”>
         <xs:element name=”knot” type=”knot”/>
         <xs:element name=”anchor” type=”anchor”/>
         <xs:element name=”tie” type=”tie”/>
      </xs:choice>
      <xs:assert test=”every $k in
        anchor/attribute/@knotRange satisfies
        knot[@mnemonic = $k]”/>
      <xs:assert test=”every $a in tie/anchorRole/@type
        satisfies anchor[@mnemonic = $a]”/>
      <xs:assert test=”every $k in tie/knotRole/@type
        satisfies knot[@mnemonic = $k]”/>
      <xs:assert test=”not(some $t1 in tie, $t2 in
        $t1/preceding-sibling::tie satisfies
        deep-equal($t1,$t2))”/>
      <xs:assert test=”not(some $t1 in tie/*, $t2 in
        $t1/preceding-sibling::node() satisfies
        deep-equal($t1,$t2))”/>
      <xs:assert test=”not(some $t1 in anchor, $t2 in
        $t1/preceding-sibling::anchor satisfies
        $t1/@mnemonic = $t2/@mnemonic)”/>
      <xs:assert test=”not(some $t1 in knot, $t2 in
        $t1/preceding-sibling::knot satisfies
        $t1/@mnemonic = $t2/@mnemonic)”/>
      <xs:assert test=”not(some $t1 in anchor/attribute,
        $t2 in $t1/preceding-sibling::attribute satisfies
        $t1/@mnemonic = $t2/@mnemonic)”/>
   </xs:complexType>
   <xs:element name=”schema” type=”schema”/>
</xs:schema>
```

Figure 2: XML Schema definition of an anchor schema, derived from the basic notions of Anchor Modeling presented in [].

```
<?xml version="1.0"?>
<schema xmlns="http://anchormodeling.com/schema">
  <knot mnemonic="GEN" descriptor="Gender"
      identity="bit" dataRange="string"/>
  <knot mnemonic="RAT" descriptor="Rating"
      identity="tinyint" dataRange="string"/>
  <knot mnemonic="PLV" descriptor="ProfessionalLevel"
      identity="tinytint" dataRange="string"/>
  <knot mnemonic="PAT" descriptor="ParentalType"
      identity="tinyint" dataRange="string"/>
  <anchor mnemonic="AC" descriptor="Actor"
      identity="integer">
    <attribute mnemonic="NAM" descriptor="Name"
        dataRange="string" timeRange="date"/>
    <attribute mnemonic="GEN" descriptor="Gender"
        knotRange="GEN"/>
    <attribute mnemonic="PLV"
        descriptor="ProfessionalLevel"
        knotRange="PLV" timeRange="date"/>
  </anchor>
  <anchor mnemonic="PR" descriptor="Program"
      identity="integer">
    <attribute mnemonic="NAM" descriptor="Name"
        dataRange="string"/>
  </anchor>
  <anchor mnemonic="ST" descriptor="Stage"
      identity="integer">
    <attribute mnemonic="NAM" descriptor="Name"
        dataRange="string" timeRange="date"/>
    <attribute mnemonic="LOC" descriptor="Location"
        dataRange="string"/>
  </anchor>
  <anchor mnemonic="PE" descriptor="Performance"
      identity="integer">
    <attribute mnemonic="DAT" descriptor="Date"
        dataRange="datetime"/>
    <attribute mnemonic="AUD" descriptor="Audience"
        dataRange="integer"/>
    <attribute mnemonic="REV" descriptor="Revenue"
        dataRange="money"/>
  </anchor>
  <tie>
    <anchorRole type="AC" role="parent"
        identifier="true"/>
    <anchorRole type="AC" role="child"
        identifier="true"/>
    <knotRole type="PAT" role="having"
        identifier="true"/>
  </tie>
  <tie>
    <anchorRole type="PE" role="in" identifier="true"/>
    <anchorRole type="AC" role="wasCast"
        identifier="true"/>
  </tie>
  <tie>
    <anchorRole type="PE" role="at"
        identifier="true"/>
    <anchorRole type="PR" role="wasPlayed"/>
  </tie>
  <tie>
    <anchorRole type="PE" role="wasHeld"
        identifier="true"/>
    <anchorRole type="ST" role="atLocation"/>
  </tie>
  <tie timeRange="date">
    <anchorRole type="AC" role="part"
        identifier="true"/>
    <anchorRole type="PR" role="in"
        identifier="true"/>
    <knotRole type="RAT" role="got"/>
  </tie>
  <tie timeRange="date">
    <anchorRole type="ST" role="atLocation"
        identifier="true"/>
    <anchorRole type="PR" role="isPlaying"
        identifier="true"/>
  </tie>
</schema>
```

Figure 3: An XML implementation of the XML Schema in Figure 2, based on the example in Figure 1.