

Virtual XML: A Chartered Aircraft Arrivals System

Authors: Kristoffer Rose, Susan Malaika

Here we delve into an example of the *Virtual XML Aggregator* pattern. Note that throughout this example we associate the DFDL draft's namespace URI "<http://dataformat.org/>" with the "dfdl:" prefix and, as usual, the XML Schema namespace URI "<http://www.w3.org/2001/XMLSchema>" with the "xs:" prefix.

Imagine that an airport control site receives periodic updates with all the chartered airplanes that have lifted off worldwide in the last hour, with each plane described by a number of properties:

- Originating and destination airport codes
- Time of lift-off
- Expected time of arrival
- Company that chartered the airplane
- Total number of passengers

The format, described next, is assembled out in the local control towers by a small utility that has been distributed and developed in an ad-hoc manner.

Since the information is collected per country, the information is provided to subscribers as a frequently updated zip archive with a member file per country where planes have taken off. (This format was chosen because zip archives are easy to assemble.) Each such member file is named with the ISO country code and contains records that were built using a C program with the following data structure:

```
struct take_off
{
  char[3] origin_airport;
  char[3] destination_airport;
  long lift_off_time; /* Epoch time */
  short passenger_count;
  short company_name_length;
  char[] company_name; /* company_name_length bytes using UTF-8 encoding
*/
}
```

Here is, for example, the hexadecimal dump of the above structure for two records,

- From *CPH* to *JFK* on 2005-07-23 at 06:52:02 with 345 passengers chartered by *Flywell*.
- From *CPH* to *CDG* on 2005-07-23 at 06:52:41 with 211 passengers chartered by *Supair*.

```
00000000 43 50 48 4a 46 4b 00 00 11 c8 e1 42 59 01 07 00 |CPHJFK....BY...|
00000010 46 6c 79 77 65 6c 6c 43 50 48 43 44 47 00 00 b2 |FlywellCPHCDG...|
00000020 cb e1 42 d3 00 06 00 53 75 70 61 69 72          |..B....Supair|
```

Virtual XML – Chartered Aircraft Arrivals Example

Assuming the information in the two records is stored in a file "DK" (because the CPH airport is in Denmark) and then compressed into a zip file with just this entry. Such a zip archive looks as follows:

```
00000000 50 4b 03 04 0a 00 00 00 00 00 ea 36 f7 32 8b 01 |PK.....6.2..|
00000010 c8 a1 2d 00 00 00 2d 00 00 00 02 00 15 00 44 4b |...-...-.....DK|
00000020 55 54 09 00 03 b7 cd e1 42 b8 ca e1 42 55 78 04 |UT.....B...BUx.|
00000030 00 e8 03 e8 03 43 50 48 4a 46 4b 00 00 11 c8 e1 |.....CPHJFK.....|
00000040 42 59 01 07 00 46 6c 79 77 65 6c 6c 43 50 48 43 |BY...FlywellCPHC|
00000050 44 47 00 00 b2 cb e1 42 d3 00 06 00 53 75 70 61 |DG.....B....Supa|
00000060 69 72 50 4b 01 02 17 03 0a 00 00 00 00 00 ea 36 |irPK.....6..|
00000070 f7 32 8b 01 c8 a1 2d 00 00 00 2d 00 00 00 02 00 |.2.....-...-.....|
00000080 0d 00 00 00 00 00 00 00 00 00 a4 81 00 00 00 00 |.....|
00000090 44 4b 55 54 05 00 03 b7 cd e1 42 55 78 00 00 50 |DKUT.....BUx..P|
000000a0 4b 05 06 00 00 00 00 01 00 01 00 3d 00 00 00 62 |K.....=...b|
000000b0 00 00 00 00 00 |.....|
```

(In this case the file is so short that no compression was done but just the packing -- you can see the member name DK at the end of the second line.)

Queries

Furthermore we assume that the user wants to ask some questions such as:

- How many passengers have the latest update put in the air?
- What are the destinations of the planes that left from CPH airport within this period?
- How many passengers are expected to arrive in JFK from planes that lifted off in this period?

Using Virtual XML in the Chartered Aircraft Arrivals Use Case

We will show that these queries can be very elegantly and efficiently processed using the *Virtual XML* methodology.

The Idealized XML Schema

The first thing we'll need is an XML Schema that expresses the perfect XML view on our data. Here is a proposal.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:fu="http://www.example.com/xml/virtual/sample/flight-
update"
targetNamespace="http://www.example.com/xml/virtual/sample/flight-
update">
  <xsd:element name="flight-updates">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="country" type="fu:Country"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Virtual XML – Chartered Aircraft Arrivals Example

```
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:simpleType name="Airport">
  <xsd:restriction base="xs:string" length="3" pattern="[A-Z][A-Z][A-Z]"/>
</xsd:simpleType>

<xsd:complexType name="Country">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="flight-update" type="fu:Flight-Update"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xs:string"/>
</xsd:complexType>

<xsd:complexType name="Flight-Update">
  <xsd:sequence>
    <xsd:element name="origin" type="fu:Airport"/>
    <xsd:element name="destination" type="fu:Airport"/>
    <xsd:element name="lift-off-time" type="xs:dateTime"/>
    <xsd:element name="passenger-count" type="xs:short"/>
    <xsd:element name="company-name" type="xs:string"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Notice how we have chosen to use elements almost exclusively (the exception being the country element's name attribute). With this schema we can now express the queries in XPath.

How many passengers have the latest update put in the air?

```
sum(//passenger-count)
```

What are the destinations of the planes that left from CPH airport within this period?

```
sort(//flight-update[origin='CPH']/destination)
```

How many passengers are expected to arrive in JFK from planes that lifted off in this period?

```
sum(//flight-update[destination='JFK']/passenger-count)
```

The Virtual XML Pipeline

With our target schema now fully defined we can present a strategy for solving the problem with *Virtual XML*. We shall base everything on XQuery with two additional functions:

- `unzip(zip-url)` returns an XML version of a zip archive in *zip-url* with all members decompressed (explained below).
- `dfdl(dfdl-url, base64value)` returns an XML document generated from the binary data in *base64value* using the DFDL specification in *dfdl-url*.

Interpreting the Zip archive structure

Our collection of *Virtual XML* views includes an adapter for zip archives. It generates a *Virtual XML* view of a zip file which is valid for the following XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:zip="http://ibm.com/xml/virtual/Zip"
            targetNamespace="http://ibm.com/xml/virtual/Zip">

  <xsd:element name="zip">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="entry" type="zip:Entry"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="Entry">
    <xsd:sequence>
      <xsd:element name="bytes" type="xs:hexBinary"/>
    </xsd:sequence>
    <xsd:attribute name="comment" type="xs:string"/>
    <xsd:attribute name="compressed-size" type="xs:long"/>
    <xsd:attribute name="crc" type="xs:long"/>
    <xsd:attribute name="extra" type="xs:hexBinary"/>
    <xsd:attribute name="method" type="xsd:long"/>
    <xsd:attribute name="name" type="xs:string"/>
    <xsd:attribute name="size" type="xs:long"/>
    <xsd:attribute name="time" type="xs:dateTime"/>
  </xsd:complexType>
</xsd:schema>
```

With this encoding, our Zip archive from before is expressed as the XML

```
<?xml version="1.0" encoding="UTF-8"?>
<zip:zip xmlns:zip="http://ibm.com/xml/virtual/Zip">
  <zip:entry name="DK" size="45" time="2005-07-20T06:55:20Z">

    <zip:bytes>Q1BISkZLAAARYOFCWQEHAEZseXdlbGxDUEhdREcAALLL4ULTAAYAU3VwYWly
  </zip:bytes>
  </zip:entry>
</zip:zip>
```

(the bytes element contains the base64 encoded concatenation of the two entries).

Interpreting the flight update C structure

Here is an adapter written in DFDL to parse the binary struct take_off member lists into an appropriate substructure:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Virtual XML – Chartered Aircraft Arrivals Example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fu="http://www.example.com/xml/virtual/sample/flight-
update"
targetNamespace="http://www.example.com/xml/virtual/sample/flight-
update">

  <xsd:annotation><xs:appinfo source="http://dataformat.org/">
    <dfdl:definitions>
      <dfdl:use configuration="binaryTypes"/>
    </dfdl:definitions>
  </xs:appinfo></xsd:annotation>

  <xsd:element name="root">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="flight-update" type="fu:Flight-update"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:simpleType name="Airport">
    <xsd:restriction base="xs:string" length="3" pattern="[A-Z][A-Z][A-Z]"
      dfdl:characterSet='US-ASCII' />
  </xsd:simpleType>

  <xsd:complexType name="Flight-update">
    <xsd:sequence>
      <xsd:element name="origin" type="fu:Airport"/>
      <xsd:element name="destination" type="fu:Airport"/>
      <xsd:element name="lift-off-time" type="xs:dateTime"
dfdl:dateFormat="epoch"/>
      <xsd:element name="passenger-count" type="xs:short"/>
      <xsd:element name="company-name" type="xs:string"
dfdl:characterSet='UTF-8'
dfdl:byteOffset="18" dfdl:byteSize="{@company-name-length}"/>
    </xsd:sequence>
    <xsd:attribute name="company-name-length" type="xs:short"
dfdl:byteOffset="16"/>
  </xsd:complexType>

</xsd:schema>
```

Notice that

- Most elements are generated in a form that directly fits our target schema.
- The lift-off-time is interpreted in "Epoch" form, *i.e.*, as the number of seconds since January 1, 1970.
- We explicitly indicate the start offset (relative to the parent) of the company-name-length attribute and company-name strings because attributes do not have a natural sequence.

Virtual XML – Chartered Aircraft Arrivals Example

We will assume that this DFDL format specification is saved as <http://ibm.com/xml/virtual/sample/flight/update.dfdl> and we will use the value in the dfdl function below.

Building the result instance

We can express the construction of the idealized form of the XML document using a simple XML Query function of the following form:

```
import schema default element namespace
"http://www.example.com/xml/virtual/sample/flight-update";

declare namespace v = "http://ibm.com/xml/virtual";
declare namespace zip = "http://ibm.com/xml/virtual/Zip";

declare variable $format :=
"http://www.example.com/xml/virtual/sample/flight/update.dfdl";

declare function collect-flight-updates($zip as xs:string) as document-
node()
{
  document
  {
    element flight-updates
    {
      for $entry in v:unzip($zip)//zip:entry
      return
      element country
      {
        attribute name { $entry/@name }
        for $update in v:dfdl($format,
data($entry/zip:bytes))//flight-update
        return
        element flight-update
        {
          $update/*
        }
      }
    }
  }
}
```

Notice how:

- the entries of the zip archive are mapped directly to corresponding country element names in the flight-update namespace,
- the dfdl function is used to decode the archive member for each country which is then traversed to get the actual flight updates,
- we explicitly recreate the fu:flight-update element to ensure that only the child elements are copied in to filter out the DFDL-generated "helper" attributes.

All these properties of the query allows the view generated by the query to be virtual in the sense that navigating the view can be done without materializing the entire query result.

Virtual XML – Chartered Aircraft Arrivals Example

In this context we can then simply call the collect-flight-updates function

```
declare variable $latest-zip :=  
"http://www.example.com/xml/virtual/sample/dk.zip";  
collect-flight-updates($latest-zip)
```

to obtain

```
<fu:flight-updates  
fu="http://www.example.com/xml/virtual/sample/flight-update">  
  <fu:country name="DK">  
    <fu:flight-update>  
      <fu:origin>CPH</fu:origin>  
      <fu:destination>JFK</fu:destination>  
      <fu:lift-off-time>2005-07-23T04:52:02Z</fu:lift-off-time>  
      <fu:passenger-count>345</fu:passenger-count>  
      <fu:company-name>Flywell</fu:company-name>  
    </fu:flight-update>  
    <fu:flight-update>  
      <fu:origin>CPH</fu:origin>  
      <fu:destination>CDG</fu:destination>  
      <fu:lift-off-time>2005-07-23T04:52:41Z</fu:lift-off-time>  
      <fu:passenger-count>211</fu:passenger-count>  
      <fu:company-name>Supair</fu:company-name>  
    </fu:flight-update>  
  </fu:country>  
</fu:flight-updates>
```

where <http://www.example.com/xml/virtual/sample/dk.zip> is the URL for our small sample zip archive. The spacing of the XML above, as usual, is just a way to make the structure clear.

But the most interesting aspect is that the new function is simple enough that it in turn defines a new *Virtual XML* view so we can add it to our repertoire. We can now compute the queries.

```
sum(collect-flight-updates($latest-zip)//passenger-count) :  
  computes how many passengers the latest update has put in the  
air: 556.
```

```
sort(collect-flight-updates($latest-zip)//flight-  
update[origin='CPH']/destination) : computes and sorts the destinations  
of the planes that left from CPH airport within this period: the  
sequence (CDG, JFK).
```

```
sum(collect-flight-updates($latest-zip)//flight-  
update[destination='JFK']/passenger-count)
```

computes how many passengers are expected to arrive in JFK from planes that lifted off in this period: 345.

Virtual XML – Chartered Aircraft Arrivals Example

These numeric results are much smaller in size than the full example XML and thus illustrate how we can benefit from virtualization by not materializing the whole XML version but merely exploiting that the composition language, here XQuery, lives in the same XML context as the virtualization components, in this case the unzip and dfdl functions.
