DFDL Working Group, December 2004 Face to Face Meeting Summary

Status of This Memo

This memo provides information to the Grid community regarding the status and progress of the working group in its objectives. This document does not define any standards or technical recommendations. Anyone may read this document; however, it's purpose is for internal communications of the working group..

Abstract
This document is a status report of a face-to-face meeting of the WG held in Los Gatos, California, USA on 2004-12-06 and 2004-12-07.

Revision History

Latest entry ***at the top*** please			
Version	Author/Contributor	History	Date(yyyy-mm-dd)
002	Jim Meyers	Edited, clarified some points. Added some questions	2004-12-14
001	Mike Beckerle	Created.	2004-12-08

Contents

1 Introduction

The DFDL WG held a face to face meeting, hosted by Ascential Software at their Los Gatos California, USA offices. The meeting was held on 2004-12-06 and 2004-12-07 and ran all day both days including lunch and dinner.

Attendees:

Jim M~~e~~yers - PNNL

Alan Chappell - PNNL

Martin Westhead - Independent Consultant

Suman Kalia - IBM

Mike Beckerle - Ascential Software

2 Open Issues List

At the conclusion of the meeting Tuesday evening, we created this list of open issues:

- composability and extensibility - XML syntax support vs. flexibility
- common rep model and unified list of properties and their meanings
- scoping issue. What properties are allowed where, and how they combine when there are multiple definitions in scope
- discontiguous representations - need examples and to decide whether and how to handle this case. One example is a record containing variable length strings where all the length fields ~~proceed~~precede all the variable-length content pieces.
- hidden layer elements - we agreed on these for filtering source and target streams; however, we didn't conclude discussion about whether ordinary elements can be hidden from the logical model
- top down: given an XSD predefined, what is needed to write it out in binary?
- check (periodically) on any w3c activities around adding a multi-dimensional array construct to XML so we can influence this if needed.

No doubt the above list is incomplete.

The composability (compositionality?) and extensibility issue is worth further elaboration:

2.1 Composability and Extensibility

We spent ~~a~~lot of time at the F2F on this issue. We generally agree that (1) DFDL must be extensible to allow unanticipated formats to be handled (2) it is desirable if possible that this be seamless, I.e., an extension is not distinguishable from a "built-in" capability (3) an extension can be highly performant (4) an extension can be a "black box" add on of external code, or can be a white-box composition of DFDL features which is named and labeled so that it looks seamless. Of these requirements, number 1 is critical, number 4 is critical, and 2 and 3 are highly desirable, but perhaps less critical to achieve. That is, extensibility is critical, but some syntactic burden is potentially acceptable depending on trade-offs with other desirable characteristics for DFDL.

There is a strong tension between our desire to create a clean embedding of DFDL into the XML Schema Description language (XSD), and the ability to have extensibility and flexibility. XSD isn't very good at this, and other XML-embedded languages such as the ANT build-automation language do not take strong advantage of XML schema validation to insure language correctness.

Suman showed how the current IBM Websphere BI approach obtains good leverage from XML schema validation to provide a good measure of checking for the user. This technique depends on this XSD "trick":

```
<element name="..." type="TYPENAME">
  <annotation><appinfo source="...">
    <TYPENAME_ELT ...>
          ....representatio properties....
    </TYPENAME_ELT>
  </appinfo></annotation>
</element>
```

In this XSD fragment, the annotation contents can be validated, and if the author insures that the TYPENAME and TYPENAME_ELT are in proper correspondence (which the validator can't check), the XML validator will at least insure that the representation properties are sensible and allowed for this type.

~~We were unable to quickly figure out a way~~An alternative, which attempts to ~~to~~ reconcile this technique with extensibility desires, was discussed. In this approach, the <TYPENAME_ELT…> is parameterized by separating the logic of reading the value from the type itself, i.e. a type descriptor contains an element representing the reading algorithm which then includes relevant parameters. Since the element representing the reading algorithm can be strongly typed (e.g. required to be an extension of a base type), ~~. E.g.,~~ an open-ended set of user-extensible representation parameters ~~is fundamentally in tension with a~~ can still be validated against a static XML Schema created by including the new algorithm type definition in the ~~for~~ DFDL. Essentially this approach recognized that two forms of extension exist – creating new types and creating new ways to read types from underlying representations. Whether this model, which requires management of both type and algorithm hierarchies, is worth the added complexity is still in debate. Compromises in this area where built-ins are better checked than extensions ~~may~~ might be ~~needed~~ a reasonable alternative and we need a concrete set of use-cases which illustrate the extensibility concerns in order to move forward.

We were also not in general agreement as to whether one should be able to put irrelevant representation properties on an element. Some of us thought this would be misleading. E.g., a user could spend time tweaking a bunch of properties when trying to debug a data format definition only to eventually realize that they weren't even being used by that particular representation. Other people believe the ability to put consistent sets of properties which aren't specific to a data type would be convenient. E.g., on an integer element, a charset property indicating "ebcdic" in a binary format wouldn't be used, but should be silently ignored, and a byte-order property on a string should similarly be silently ignored.

We all noticed that the OMG TD model and the description of "transforms" by Alan Chappell had marked similarities, but we weren't able to drive them to a precise conclusion. There is more work to be done here, but the notion arose that a TD and a "transform" may be the same or are at least ~~very~~ closely related (i.e. transforms represent a parameterized type model as discussed above).

3 Agreed Items

Below are items about which we had some agreement, and discussion of them.

3.1 DFDL's Goals

We agreed that DFDL's design goals are about solving the data representation problem, and not the entirety of the data mapping problem from source format to an arbitrarily different target format. That is, while DFDL describes a logical format for data, and then a mapping of a physical representation to that logical model, ~~there needs~~DFDL is intended for use when there is ~~to be~~ some degree of structural similarity between the logical and physical models. Terms like "reasonably compatible" were used to describe this relationship. Whether DFDL can be used in more complex cases (e.g. to natively represent in DFDL itself the transformations involved in zip decompression) is outside the scope of the standards discussions.

The concept of "reasonably compatible" seems to be about controlling the complexity of the annotations of the elements of an XSD needed to be to reflect the representation. Without depending on external black-box extensions, there is a goal that DFDL should be able to describe

the physical-representation to logical model mappings that can be expressed without introducing the need for annotations which contain any sort of iteration construct.

While we didn't agree that DFDL's white-box extension/annotations wouldn't contain iteration, we did agree that we wanted to understand what could be expressed without an iteration construct so that we can determine if this is sufficient.

3.2 OMG Type-descriptor Model

We examined the OMG type-descriptor model, and while we were not in agreement about how best to cast this into our current XSD-based approach to DFDL, we did have general agreement that this model provides a valuable and carefully chosen set of properties for describing binary data and we should leverage these definitions as much as possible. There is much overlap between this OMG TD model and other materials contributed by Ascential, and an effort combine ideas here is needed.

3.3 Avoiding XSD inside the Annotations

We advanced the hypothesis that we can avoid any need to have XSD syntax nested inside DFDL annotations. The trick is to create a top-level type definition and refer to it by name from inside the annotations. We need to look at every place that we have the potential for nesting XSD inside our annotations and see if this hypothesis can hold and revisit if it is in fact desirable to restrict our annotation syntax in this way, or whether this should be considered a matter of style. Layered models are the tricky issue here. Having to create separate top-level type definitions every time may be too clumsy.

3.4 XSLT contribution

We looked at XSLT briefly. Some of us like XSLT, some dislike it mostly because it is too powerful and potentially very hard to implement for high-performance. Some of us like XQuery better than XSLT. The role of either of these languages in the DFDL spec was unclear except to the extent that we agree that very powerful languages like these should be avoided in DFDL if possible since they add to implementation and end-user complexity.

We did agree that there are some useful XML style idioms in XSLT E.g., they use this idiom

```
<xsl:value-of name="variable" select="...."/>
```

where you always have the alternative to write the same thing as:

```
<xsl:value-of name="variable">....</xsl:value-of>
```

That is, one can always choose to put the content of the value-of tag into an attribute value or the element contents.

3.5 I/O Symmetry

We agreed that it is strongly desirable in many use cases for a DFDL description to allow both reading and writing a data format. However, we acknowledge that because the reading process may not preserve all the information content of the input data, it may not be possible for a DFDL-based application program to read and write the exact same data file. For example, a DFDL description should be able to express things like comment syntax in text representations and strip it out of the data; ~~hwoever~~however, this implies those comments are not really part of the logical data model and so would not be recreated if the "same data" were read in and then written back out. So there are acknowledged limits on the I/O symmetry issue. However, it is very desirable for any DFDL description which preserves all the information content of an input file to be able to recreate the exact same file on output. Based on this we concluded that all built-in representation types should support both input and output of data in that representation. Use of choices and runtimeValue expressions can create non-invertible input read conversions, and constructs must be provided allowing a DFDL author to insert corresponding writer conversions so as to allow such data formats to be I/O symmetric.

Comment [JDM1]: What does this mean for the DFDL language? Just that we don't want to introduce features that make this hard/impossible? Or is this a remark about DFDL parser implementation? If so, we should make that clear and I'm not so sure we agree – could I build a high speed DFDL parser and a separate DFDL inverter and be compliant?

3.6 Kinds of Operations in DFDL (filters, readers, writers, functions)

In an attempt to make progress discussing composability issues, ~~wW~~e ~~concluded that there are~~explored describing DFDL in terms of ~~4~~ kinds of operational entities ~~in DFDL~~ that we had previously been lumping under the single confusing name of “transforms”:

- reader (or read conversion)
- writer (or write conversion)
- filter
- function

These have different signatures with respect to types.

Let T0, T1,... be types. E.g., character or byte

Let stream<T> be a stream of elements of type T. You can think of a stream<T> as a ~~fixed length or~~ potentially unbounded array containing elements of type T, plus a current-position pointer value within that array. A file and a position in it when taken together are what we mean by a stream of bytes for example.

Given these, we can give the signature of a reader:

reader: stream<T0> -> T1, stream<T0>

That is, a reader takes a stream<T0>, which is called a “source” and produces a element of type T1, and a shorter stream<T0>. The T1 value is computed by reading some of the T0 elements of the stream.

A writer has this signature:

writer: stream<T0>, T1 -> stream<T0>

That is, it takes a stream<T0>, and an item of type T1, and puts the representation of the T1 (in terms of the T0 units) adding them onto the input stream to produce the output stream, which is called the target stream.

These concepts provide a mathematical way to model the semantics of cursors on input files.

A filter’s signature:

filter: stream<T0>->stream<T1>

That is, a filter changes a stream. E.g., a byte stream can become a character stream by way of a filter that implements a character set encoding. If T0 = T1, then the filter preserves the stream type, and this is a useful special case. E.g., removing blank lines from a text input is such a special case.

A function is not involved with streams:

function: T0->T1

E.g., addition is a function, and clearly in DFDL one can add the values of elements to provide values of other elements.

A hypothesis is that one can express the semantics of a DFDL descriptor by describing how it is equivalent to a network of filters, readers, writers, and functions. It is recognized that some sort of iteration scheme is needed here to cope with arrays in the DFDL description. Further, one might wish to encapsulate a portion of the graph as a reusable element. Whether it will be possible to limit such encapsulation in DFDL, such that subgraphs are always representable as composite readers, writers, filters or functions (or some subset of those), and still meet the extensibility goals is a subject for futher work.

An element of a DFDL schema has several ways it can get its value. Most commonly a reader produces the value of the element, and a writer outputs the value of the element. Optionally, one can specify from what source the reader for that element takes its input, and to what target the writer for the element sends its output. The reader and writer can be either implicit (typical, based on type and representation properties), or explicitly specified.

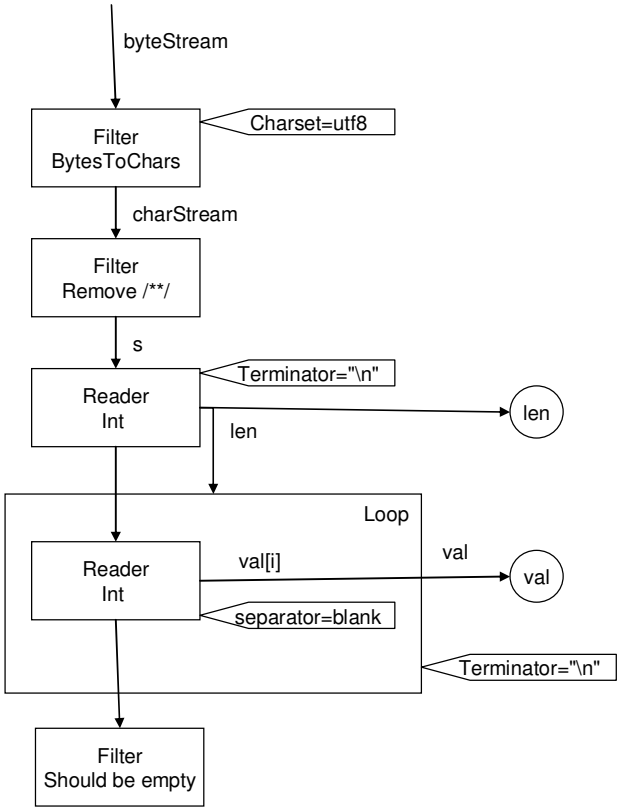
In addition, any source or target can have filters attached to it.

Finally, some types of elements (strings and byte arrays at least, probably others) can be converted into streams, so that one element can indicate that its source stream is the value of another element. This allows a very powerful kind of layering. The DFDL fragment below illustrates how this is used to handle a file format where filters are used to strip-out C-style comment syntax and then to describe a variable-length array format:

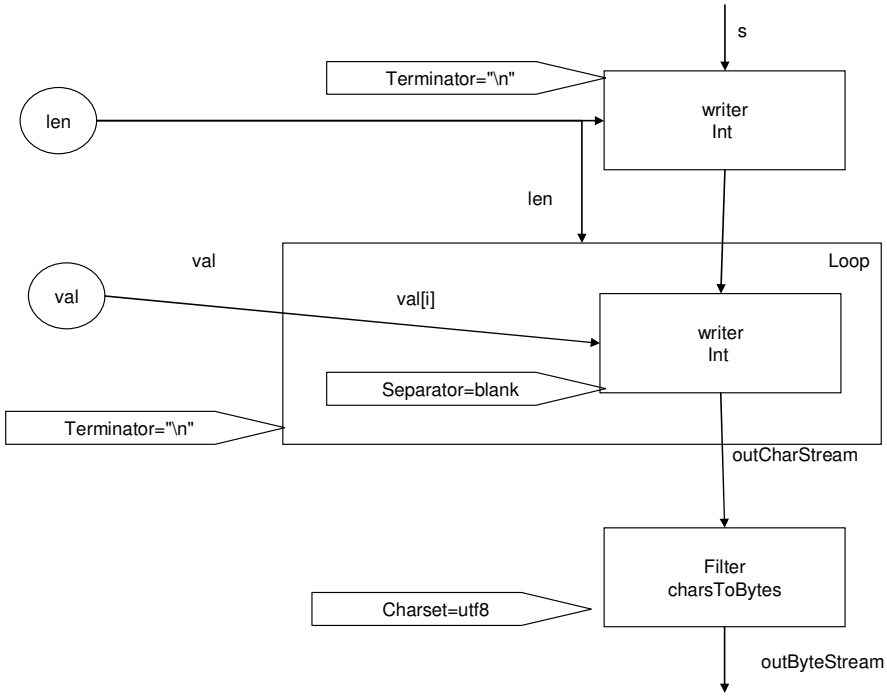
```
<element name="charstream" type="dfdl:sourceStream">
  <annotation><appinfo source="...">
   <dfdl:sourceStreamTD>
    <charset>utf-8</charset>
    <source>byteStream</source>
    <filter>bytesToChars</filter>
   </dfdl:sourceStreamTD>
  </appinfo></annotation>
</element>

<element name="s" type="dfdl:sourceStream">
  <annotation><appinfo source="...">
   <dfdl:sourceStreamTD>
    <filter>replaceRegexp("...regexp for C-comments...", "")</filter>
    <source>charstream</source>>
   </dfdl:sourceStreamTD>
</appinfo></annotation>
</element>

<element name="t" type="dfdl:targetStream">
  <annotation><appinfo source="...">
   <dfdl:targetStreamTD>
    <charset>utf-8</charset>
    <target>outbyteStream</target>
    <filter>charsToBytes</filter>
   </dfdl:targetStreamTD>
  </appinfo></annotation>
</element>

<element name="toplevel">
  <annotation><appinfo source="...">
   <dfdl:instanceTD>
    <source>s</source>
    <target>t</target>
    <repType>text</repType>
   </dfdl:instanceTD>
  </appinfo></annotation>
  <sequence>
      <element name="len" type="int">
         <annotation><appinfo source="...">
           <intTD>
            <terminator>\p{newline}</terminator>
           </intTD>
         </appinfo></anntation>
      </element>
      <element name="val" type="int" minOccurs="0" maxOccurs="unbounded">
         <annotation><appinfo source="...">
            <intTD>
              <arrayTD>
                <storedLength>../len</storedLength>
                <terminator>\p{newline}</terminator>
                <separator>\p{space}</separator>
              </arrayTD>
              <numbase>10</numBase>
              <reader name="myIntReader">
```

```
                    <numberOfBits>13</numberOfBits>
                  </reader>
                </intTD>
              </appinfo></anntation>
            </element>
          </sequence>
        </element>
```

We drew a box-and-arrows diagram showing how this DFDL corresponds to a flow-graph of filters and readers for data input, and we drew a similar corresponding flow-graph of writers and filters for data output. This notion that there is a flow-graph that is implied by a DFDL descriptor appears to be a powerful way to capture the semantics of a DFDL description.



The above is the input diagram showing filters and readers. The representation properties are shown in the block arrows attached to the filters and readers. The small circles are the elements of the logical data model.



The above shows the output side flow diagram.

4 Action Items

We concluded that our plans to have an internal group working draft of our standard in time for GGF13 in March 2005 were unrealistic, however we believe we can make progress on these two items in time:

1. update primer with more and larger examples. Bring up to date with latest thinking. (doing this will force us to confront some of the open issues mentioned above. E.g., scoping)
2. merge the OMG TD model contribution with the other rep-properties contributions to form a unified model of all the properties.

Roughly: Mike and Suman have #2 as they are the contributors of this material. Martin, Jim, and Alan have the ball on item 1.