Some Notes of comparison between DFDL and Defuddle

Alejandro Rodriguez, Robert E. McGrath

30 September 2010

1. Introduction

These notes are an analysis of the OGF Data Format Specification Language (DFDL), in public comment this spring [1]. The DFDL defines a language to describe non-XML data, which is embedded in annotations in an XML Schema (XSD) [10]. The DFDL defines both a data model (which is a subset of the XML Infoset [2]) and an abstract parser [1].

In earlier work, the Defuddle parser implemented elements of the DFDL standard as it was defined at the time [5, 6, 8, 9]. In addition to implementing a version of the DFDL, Defuddle has explored semantic extensions [4, 7]. Defuddle is available at:

http://sourceforge.net/projects/defuddle/

With the release of the draft standard, it is now necessarily and possible to analyze the Defuddle software to discern how it matches the Version 1 specification. This note summarizes our analysis.

DFDL Conformance

The DFDL Working Group has not formally defined "conformance" for the specification, nor produced normative test cases to date.

Since DFDL is conceptually similar to XML Schema [10], conformance requirements will probably be similar. Conformance for XSD is a complex issue, involving behavior of parsers, platform dependent data representations, and reference to other standards (e.g., URI, XPath [3], XML [11], etc.).

Our conclusion is that, at this time, there is no clear definition of formal conformance. Therefore, we cannot formally assess Defuddle's conformance to the DFDL standard. This document is a best effort to informally assess conformance.

DFDL Implementations

There are no announced implementations of the DFDL standard. The working group has been creating one or more implementations as part of its work, and partial implementations have been discussed in the past. However, the draft standard has been released for comment without any identified reference implementation.

Therefore, it is not possible to directly compare Defuddle to any other DFDL implementation at this time.

2. Analysis of DFDL and Defuddle

This section discusses specific points where Defuddle does not (or may not) agree with the DFDL specification.

2.1. Logical Parser Requirements: Push-back parser

The DFDL specification provides a push back parser. Specifically, the DFDL logical parser is defined to be a "recursive-descent parser with guided, potentially unbounded lookahead" (DFDL, section 9.1).

There are at least two levels where this functionality comes into play:

1) Uncertainty points:

These correspond to XSD constructs 'choices' {cite XML standard?} and a special DFDL construct called 'unordered sequences.' Choices allow the parser to branch between several possible options for the next element, and backtracks as necessary if a given option cannot be parsed, whether this is discovered immediately or after parsing several nested elements. Unordered sequences are a mixed between XSD:choice and XSD:all, allowing an element to contain one or more of a group of nested sub-elements.

PROVIDED BY DEFUDDLE: Defuddle implements the behavior for choices.

It is unknown if Defuddle currently provides unordered sequences, but it seems plausible to implement it in the current architecture.

2) Terminators/separators, last element in a array:

Although not explicitly stated in the specification, some examples of IBM DFDL implementation show cases like this:

Given data of the form

c:/folder/folder/folder/file

the goal is to parse into an XML model something like this:

```
<drive>
  <folder/>
  <folder/>
  <folder/>
  <file/>
</drive>
```

The DFDL annotation would like to express this relationship with a grammar where 'folder' is defined in DFDL as an element with a post-fix separator '/'. DFDL is capable of

differentiating a file_name from a folder_name after noticing the missing '/' after the file_name. This would be represented in a grammar such as:

```
FOLDER_LIST := folder_name '/' FOLDER_LIST | file_name
folder_name := [a-z]+
file_name := [a-z]+
```

PROVIDED BY DEFUDDLE: Defuddle does not implement this capability for parsing input data. Given that Defuddle is built on top of an XML parser (i.e., DOM {cite} (?)) , it is not clear how it could be extended to use multiple look-aheads or push-back at points other than XSD:choice.

Summary: Defuddle implements the semanitcs of XSD "choice", and could probably be extended to implement the DFDL extension "Unordered sequence". Defuddle does not implement recursive parsing for data, and it would not be easy to extend it to do so because the DOM parser is not designed to support it.

2.2. Document Tree Addressing of Comment Blocks

The DFDL language is embedded in XSD annotations. Certain DFDL constructs (such as layers and variables) define elements that need to be addressed via Xpath names. For example,

The Xerces implementation of DOM creates a tree of Java objects corresponding to the XML information model. XPaths refer to objects in this tree, which can be relative (i.e., to '.', the current node).

However, XML annotations and comments are implemented as nodes, but the nodes do not have a location in the tree of java objects. Since these objects do not appear in the output, they are not normally involved in Xpaths.

Since all DFDL constructs are in annotations, they have no path (i.e., they cannot be addressed within the java code) and also cannot be related to relative paths (i.e., they cannot successfully use relative paths in the Java code).

The upshot is that several features cannot be completely implemented in Defuddle, including:

- Hidden layers (section 2.5) – cannot be addressed
- Variables and computed values (section 2.4) cannot be addressed

2.3. Unparsing

"Unparsing" means translating from XML into a non-XML data file, as defined by a DFDL schema. I.e., the reverse of parsing as described above.

DFDL allows translating an XML document to a format described by a DFDL schema, and includes several constructs to specify the behavior during unparsing, such as default values, whether to include last terminator, what to do with empty elements, etc.

PROVIDED BY DEFUDDLE: Defuddle does not implement unparsing, nor the constructs used to control unparsing. We have not evaluated how (or whether) unparsing could be done in Defuddle, so the scope of work is unknown but suspected to be very large.

2.4. Variables

DFDL includes user-defined variables and mechanisms for assigning values or using them in XPath expressions.

PROVIDED BY DEFUDDLE: Variables are partly implemented in Defuddle. Defuddle includes some pre-defined variables, such as 'counter'.

However, due to the inability to address them (see 2.2 above), they cannot be used for most cases.

2.5. Hidden nodes

In DFDL, an XML element can be defined inside an annotation as hidden. The element is ignored by XSD validators and is not output as part of the parsed XML document. However, it is parsed and considered as present during the processing of the original input file.

PROVIDED BY DEFUDDLE: Yes, this is equivalent to the 'representation layer' construct in Defuddle.

2.6. Assertions, Discriminators

These are XPath expressions evaluated before or after parsing an element that can be used as validation or to guide the parser in points of uncertainty.

PROVIDED BY DEFUDDLE: Probably not. Conceivably, these could be implemented, provided that Defuddle is enhance with a full implementation of XPath (see note at the end).

2.7. Trimming, padding

DFDL allows to specify padding characters and trimming to left, right or center both in text and binary modes.

PROVIDED BY DEFUDDLE: No. It could reasonably be added.

2.8. Empty elements, defaults

The DFDL defines the behavior of empty elements and defaults.

PROVIDED BY DEFUDDLE: No.

2.9. Terminators, separators

The DFDL specification defines syntax to describe initiators, separators, and terminators; and also specifies the behavior of the parser.

PROVIDED BY DEFUDDLE: Defuddle implements a different syntax and behavior.

2.10. Validation

DFDL compliant software is supposed to validate generated XML against their schemas, using the different XSD facilities to specify allowable values. It is unclear whether this validation has any effect on the parsing (would finding an illegal value trigger a push-back?).

PROVIDED BY DEFUDDLE: Probably not. However, it is trivial to add a posteriori validation.

Note: Interestingly, there is not 'unordered sequences' in XSD. There is a similar construct called 'all'. Documents that contain unordered sequences will not validate against their schema by a regular XML validator, unless DFDL reorders the elements before output, but all elements would still be required, so why not just use 'all' ('all' is not supported by DFDL)?

2.11. Recursion

Neither DFDL nor Defuddle (possibly?) provides for elements defined recursively. This seems an important omission since many common data formats allow recursion (YAML, XML itself). There is nothing in the syntax of the language that would have to be added, and since the DFDL specification does not include a formal semantics nor parsing algorithms, one is left to wonder why recursion was specifically excluded.

2.12. Expression evaluation and Xpath

Note: Currently Defuddle uses an in-house, partial implementation of XPath to evaluate expressions. Eventually a fully compliant, freely available XPath engine could be incorporated.

3. Summary and Conclusion

The Defuddle parser implements many of the concepts of the DFDL draft standard, though it does not follow the syntax closely. Many of the features of the DFDL specification are only partly implemented.

In addition, the Defuddle parser itself suffers from deficiencies due to the JAXME parser that it builds on.

Defuddle was a very useful and successful experiment, especially as a platform for exploring semantic extensions [4, 7]. However, it would take substantial effort to bring it into conformance with the DFDL draft specification. Given the known problems with the Defuddle parser and JAXME, it is not clear that it would be worth the effort to try to modify Defuddle.

4. Acknowledgements

This work was supported through National Science Foundation Cooperative Agreement NSF OCI 05-25308 and Cooperative Support Agreements NSF OCI 04-38712 and NSF OCI 05-04064by the National Archives and Records Administration.

5. References

1. Beckerle, M., M. Westhead, J. Myers, S. Kalia, S. Hanson, T. Sugden, T. Talbot, R. McGrath, G. Judd, D. Sasser, D. Loose, E. Smith, K. Rose, A. Powell, S. Parker, P. Lambros, D. Glick, T. Kimber, S. Fetzer, and S. Marting, *Data Format Description Language (DFDL) v1.0 Core Specification*. Open Grid Foundation Proposed Recommendation 2010. http://www.ogf.org/Public_Comment_Docs/Documents/2010-03/draft-gwdrp-dfdl-core-v1.0.pdf
2. Cowan, John and Richard Tobin, *XML Information Set (Second Edition)*. W3C W3C Recommendation, 2004. http://www.w3.org/TR/xml-infoset
3. Draper, Denise, Peter Fankhauser, Mary Fernández, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler, *XQuery 1.0 and XPath 2.0 Formal Semantics*. W3C Working Draft, 2002. http://www.w3.org/TR/query-semantics/
4. McGrath, Robert E., *Semantic Extensions to Defuddle: Inserting GRDDL into XML*. NCSA, 2008. http://cet.ncsa.uiuc.edu/CEResources/defuddle-grddl.pdf
5. McGrath, Robert E., Jason Kastner, Alejandro Rodriguez, and Jim Myers, *Defuddle: a Tool for Format Translation and Metadata Extraction*, in *Microsoft E-science* 2009: Pittsburg.
6. McGrath, Robert E., Jason Kastner, Alejandro Rodriguez, and Jim Myers, *Experiments in Data Format Interoperation Using Defuddle*. National Center for Supercomputing Applications, Urbana, 2009. http://cet.ncsa.illinois.edu/publications/Data_Interoperation.pdf
7. McGrath, Robert E., Jason Kastner, Alejandro Rodriguez, and Jim Myers, *Towards a Semantic Preservation System*. National Center for Supercomputing Applications, Urbana, 2009. http://arxiv.org/abs/0910.3152
8. Talbott, Tara D., Karen L. Schuchardt, Eric G. Stephan, and James D. Myers, *Mapping Physical Formats to Logical Models to Extract Data and metadata: The Defuddle Parsing Engine*, in *International Provenance and Annotation Workshop*. 2006, Springer: Heidelberg. p. 73-81.

9. Talbott, Tara, Michael Peterson, Jens Schwidder, and James D. Myers. *Adapting the Electronic Laboratory Notebook for the Semantic Era*. In *International Symposium on Collaborative Technologies and Systems (CTS 2005)*, 2005.
10. W3C. *XML Schema Part 1: Structures*. 1999, http://www.w3c.org/TR/xml-schema-1.
11. W3C. *Extensible Markup Language (XML) 1.0 (Second Edition)*. 2000, http://www.w3.org/TR/2000/REC-xml-20001006.