Daffodil: A New DFDL Parser

Alejandro Rodriguez
Robert E. McGrath
30 September, 2010

Abstract

In earlier work the Defuddle parser implemented early version of the DFDL specification. With the publication of version 1 of the specification we now seek to conform to the standard. Our analysis shows that Defuddle is far from conforming to the DFDL specification, and also suffers from other shortcomings. We determined that it would be better to produce a new parser with equivalent features and much greater conformance to the standard. This parser is called "Daffodil", and it replaces Defuddle. To date, Daffodil is the only public implementation of the DFDL standard.

1. Introduction

In earlier work, the Defuddle parser implemented elements of the DFDL standard as it was defined at the time [6, 7, 12, 13]. In addition to implementing a version of the DFDL, Defuddle has explored semantic extensions [5, 8]. The Defuddle parser is available at:
http://sourceforge.net/projects/defuddle/

With the release of the OGF DFDL draft specification [2], it is now possible to consider a reference implementation. As part of the NARA project, we analyzed the OGF DFDL draft specification [11]. In this review, we evaluated the conformance of the Defuddle parser to the DFDL version 1 specification. In the course of this review, we also reviewed what is required in general to implement the DFDL specification.

From this investigation we concluded that it would be possible to implement a new DFDL parser—equivalent or better than Defuddle, and furthermore,the effort would be significantly easier than modifying Defuddle to be conformant to the specification. The new parser replaces Defuddle, and will become the NCSA implementation of DFDL. The Daffodil parser also implements the semantic extensions of Defuddle [5, 8].

2. Description of Daffodil

The new parser (christened "Daffodil") is a fresh start, based on version 1 of the DFDL specification[1]. Unlike, Defuddle it does not use JAXME [1]. Partly for this reason, the code is simpler than Defuddle.

Daffodil is built using Scala (V 2.8) [3, 9], a state of the art functional programming language which generates Java byte code (and thus is completely compatible with any Java Virtual Machine). The DFDL specification is mapped to Scala objects, the Scala language generates a

[1] Daffodil is based on Draft 42 of Version 1 of the Data Format Definition Language specification.

parser for the given schema. Daffodil builds on other open source software, but not Apache Xerces.

Like Defuddle, Daffodil is a parser generator. The input is a DFDL annotated schema and a data file. The schema is parsed and a parser for the DFDL is generated and run. The result is a DOM tree in memory (i.e., in Java data structures), which, by default, is output as XML.

Daffodil is similar to Defuddle, but simpler in several important areas. The code generation is simpler because the new parser only deals with DFDL, while Defuddle is embedded in a full XML schema generator, and, in particular, the new parser does not include full XML Schema validation except for the DFDL annotations.

The data is parsed into a DOM tree implemented with JDOM (jdom.org, [4] Ch. 14, 15). Paths are implemented with SAXEN (http://saxon.sourceforge.net/). In general, the proposed software is smaller and simpler than Xerces-based equivalents. In addition, since the overall parser is not embedded in the mass of JAXME, it will be much easier to optimize and/or replace specific data structures or algorithms (compared to JAXME).

The Daffodil parser uses Xpath to evaluate expressions as required by the DFDL specification. Defuddle used Java regexp (java.util.regex) and custom expression languages, which are similar to, but not the same as the specification. In general, the new parser will have a complete and functional path capability, so layers and computed values will work correctly (a major shortcoming of Defuddle).

The Daffodil parser does not support XML schema validata. When necessary, a schema can be validated by other tools. Also, and the XML output by Daffodil can be validated against the schema by other tools as well.

Basic Description of the Implementation

Daffodil takes a DFDL schema and produces a forest of logical parsers (collection of trees of BasicNodes), where each logical parser corresponds more or less to each component described in the DFDL schema. A logical parser (BasicNode) is a function that takes an input stream (which current position should reflect the bytes that correspond to the logical DOM element) and a context node in a DOM tree (the parent node of the XML element to be parsed) and produces the associated logical element by parsing the input and delegating to children logical parsers as needed.

Since DFDL possibly needs to evaluate XPath expressions on backward references, the tree is not built bottom-up, so a BasicNode actually adds DOM nodes to the context node before delegating to children nodes. In the case of a processing error (an error that a DFDL parser should backtrack, signaled by an ElementProcessingException), the BasicNode should cause no side effects on its arguments.

Actual reading of the input stream, including scanning of text, transformation of binary values into text values, etc., is delegated to processors (BasicProcessor and

InternalProcessor), that are responsible for understanding the physical format associated to a logical element.

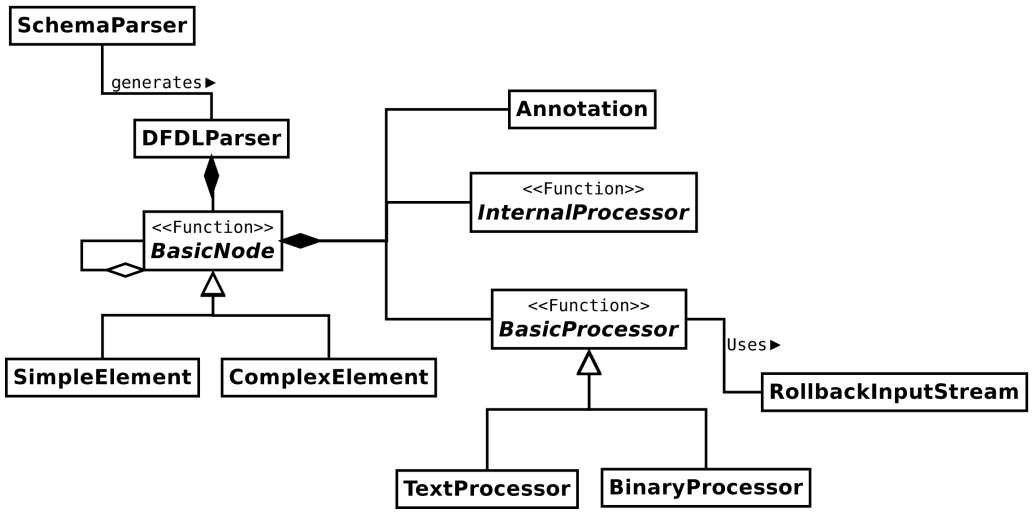

Figure 1. Main classes of Daffodil parser (from [10]).

For more details of the implementation, please see the technical note [10].

3. Summary of Improvements

The Daffodil parser has a number of improvements over the Defuddle parser. As a new parser, it is simpler and cleaner than Defuddle, and does not depend on JAXME. For these reasons, the code will be easier to change and optimize, and easier to maintain. While any DFDL parser will face challenges to scaling, the new parser will be easier to optimize and scale up.

Specific reasons to rewrite Defuddle included:

- Removes dependency in JaxMe:
  - JaxMe has been superseded by the now standard JaxB. As such, it is no longer maintained.
  - JaxMe is also fairly complex software that provides many features not needed by DFDL parsers, and the intricate way in which Defuddle interacts with it makes it difficult to perform changes to Defuddle without a deep knowledge of JaxMe itself.
  - JaxMe is essentially an XML parser and therefore lacks some features, such as speculative parsing and push/back style parsing, required by DFDL parsers. This greatly increases the difficulty of implementing a DFDL compliant parser on top of JaxMe.
  - The main objective of JaxMe, the parsing of documents into Java Beans, can be easily achieved as a post-processing step by standard and freely available tools once the documents have been converted into XML.
- Removes the source-code generation/compilation workflow:
  - Although in principle a parser compiled for a specific schema code could provide a faster parser, it is not clear that the current Defuddle implementation actually takes advantage

of this, and given the great flexibility of DFDL and the many runtime decisions a DFDL parser performs, it would be difficult to introduce this optimizations.
    - This workflow greatly increases the complexity of debugging, as errors have to be searched for in the generated code and then traced back to the generator code.
    - It increases the complexity of deployment, as it requires careful placement of temporary files and dynamic loading of libraries. Currently this is performed by scripts that remain rather fragile and make future modifications to the software difficult.
- Changes the handling of layers:
    - Defuddle handles layers as entities external to the DOM tree generated from the parsed document. This adds unnecessary complexity to the processing of runtime queries (allowed by DFDL), and is the cause of several long standing bugs in Defuddle.
- Internal representation of documents during parsing should be standard DOM trees:
    - Defuddle parses documents into beans that are later serialized as XML. This increases to some extent memory usage, and also requires custom home-brew code for evaluating expressions on the document (are required by DFDL 'computed' values). Currently Defuddle uses an incomplete basic engine for performing this evaluation. Using a standard DOM representation allows the incorporation of third-party open source libraries to evaluate XPath 2 fully compliant expressions, which satisfies the requirements of DFDL.
- DFDL Annotation syntax has changed drastically since Defuddle implementation:
    - The virtues of the new DFDL syntax might be debatable, but it's certainly not worse than the previous syntax, and it has the advantage of been thoroughly documented, which facilitates the work for writers of DFDL/Defuddle schemas. Incorporating this syntax into Defuddle, however, is a major undertaking since it requires widespread changes into the JaxMe base code.
    - Any changes to the syntax are difficult to incorporate into Defuddle for the reasons stated above, which limits extensions and experimentation with DFDL.
- Parsing of large documents requires custom memory management:
    - Defuddle stores in memory the whole of the parsed document as a tree data structure. This makes unfeasible to parse large documents. A proposed solution is to save and load from disk sections of the document as required. This is a significant change on the base code.

4. Conformance of Daffodil

The OMG DFDL Working Group is in the process of defining DFDL conformance, but, to date, has not published formal conformance specification or test cases.

The Daffodil parser is being reviewed by the working group. This review indicates that Daffodil implements almost all the core specification. We are working with the working group to obtain an initial test suite.

5. Performance of Daffodil

In earlier work, we investigated the performance of the Defuddle parser. These studies found that the Defuddle parser was only moderately scalable. In addition to a need to move to current

versions of Java, we also found limitations due to memory use due to interactions of the JAXME software and the Java run time support [8].

We evaluated the performance of the Daffodil parser against the Defuddle parser, using cases equivalent to the previous tests.

5.1. Method

The performance tests were run on an Intel Core2 Quad 2.50GHz machine with 4GB RAM running Ubuntu 10.04 64 bits and the SUN JVM 1.6. with the memory limit of JVM set to 2GB. Daffodil is built with Scala version 2.8.

The parser software was:

- Defuddle (current version) http://sourceforge.net/projects/defuddle/
- Daffodil (development) (Not available yet)

The test was a conceptual replication of the measurements reported in [8]. Please refer to the earlier note for more details of the data and the method.

The tests used two logical schemas for a simple file with rows of 100 or 1000 ASCII integers. The DFDL markup is different, but the logical layout is the same for each parser. Essentially, the input and output are identical, but the details of the schema are specific to the parser.

The datasets are files with a short header, and then a variable number of rows with comma delimited ASCII integers. The size of the input is varied by including variable numbers of rows. The schemas define an "unbounded" number of rows, so the same schema can read the different size inputs.

The important difference between Dataset 1 (rows of 100) and Dataset 2 (rows of 1000) is that the former has more XML markup for the same number of data items (i.e., there is a row for every 100 elements versus every 1000). The parser must create and manage data structures for each XML element, so the former (100 per row) requires more processing and memory use by the parser.

5.2. Results

The first test used rows of 100 numbers (ASCII, separated by commas). The number of rows was varied from 1 to 1,000,000. The overall input varied from 900 bytes to 740MB. Table 1 gives the size of the input files, and the corresponding output (XML) files.[2]

[2] The XML output contains the same data plus XML tags, so it is larger than the input. This increase in size is typical for any logical translation from binary or ASCII into XML.

Table 2 shows the elapsed time for Dataset 1. The times are averaged over three runs. The variance was small, and is not shown. For each of these input cases, in general, the elapsed time was linear with the size of the file. The two parsers performed similarly for smaller files. However, Daffodil was able to parser larger files that Defuddle could not process.

Table 1. Dataset 1: 100 numbers per row

Elements (rows x items)	*Input Size (KB)*	*Output Size (KB)*
1 x 100	0.9	4.5
10 x 100	8.8	44
100 x 100	88	432
1000 x 100	879	4300
10000 x 100	8600	43000
20000 x 100	19000	86000
30000 x 100	28000	129000
40000 x 100	35000	169000
50000 x 100	43000	211000
60000 x 100	52000	253000
70000 x 100	61000	295000
80000 x 100	69000	337000
90000 x 100	78000	380000
100000 x 100	86000	422000
500000 x 100	430000	2100000
1000000 x 100	740000	4000000

Table 2. Comparison of Daffodil and Defuddle with Dataset1 (average of 3 runs)

Elements	*Daffodil time (s)*	*Defuddle time (s)*	*Speedup (Def/Daf)*
1 x 100	0	2	-
10 x 100	1	2	2.00
100 x 100	2	3	1.50
1000 x 100	3	7	2.33
10000 x 100	21	34	1.62
20000 x 100	77	65	0.84
30000 x 100	100	95	0.95
40000 x 100	171	122	0.71
50000 x 100	194	154	0.79
60000 x 100	193	181	0.94
70000 x 100	236	203	0.86
80000 x 100	245	239	0.98
90000 x 100	333	264	0.79
100000 x 100	318	295	0.93
500000 x 100	1519	DNF	-
1000000 x 100	3496	DNF	-

The second test used rows of 1000 numbers (ASCII, separated by commas). The number of rows was varied from 1 to 100,000. The overall input varied from 8,800 bytes to 740MB.
Table 3 gives the size of the input files, and the corresponding output (XML) files.

Table 4 shows the elapsed time to parse dataset 2 for various sizes. The times are averaged over three runs. The variance was small, and is not shown. In general, the elapse time was linear with the size of the file. Daffodil performed slightly worse than Defuddle for smaller files. However, Daffodil was able to parser larger files that Defuddle could not process.

Table 3. Dataset 2: 1000 numbers per row

Elements	*Input Size (KB)*	*Output Size (KB)*
1 x 1000	8.8	44
10 x 1000	88	431
100 x 1000	879	4200
1000 x 1000	8600	42000
10000 x 1000	86000	420000
100000 x 1000	740000	740000

Table 4. Time to parse Dataset2 (average of 3 runs)

Elements	*Daffodil Time (s)*	*Defuddle Time (s)*	*Speedup (Def/Daf)*
1 x 1000	0	3	-
10 x 1000	2	2	1.00
100 x 1000	3	4	1.33
1000 x 1000	21	18	0.86
10000 x 1000	375	144	0.38
100000 x 1000	3392.666667	DNF	-

5.3. Discussion

The results show that the two parsers have similar performance for small files, and scale up linearly with size of file. The slope of the scale up depends on the schema, i.e., Dataset one scaled up much slower than Dataset 2, because it produces more XML tags per input. (See [8] for more explanation of this issue.)

As the files get larger, and the parser uses more memory, Defuddle slows dramatically and fails. In comparison, Daffodil parses much larger files, and does not slow as much. The current tests did not find an upper limit on the size of file that Daffodil can parse.

Daffodil implements memory management, "paging" out data structures to keep memory available. Paging enables Daffodil to parse much larger files than Daffodil. However, paging imposes a cost, and parsing slows as the file grows.

Daffodil is somewhat slower that Defuddle for larger files (up to the point where Defuddle fails altogether). This is almost certainly due to the overhead of the paging, which enables Daffodil to handle larger files.

It should be noted that Defuddle performed substantially better in this study than in the earlier work [8]. In addition to differences in the platforms, the earlier report used

Defuddle which used Java 1.5. The current version of Defuddle uses JDK 1.6. Evidently, JDK 1.6 has substantial performance improvements over 1.5.

6. Conclusion

With the publication of version 1 of the Data Format Definition Language specification we now seek to conform to the standard. An analysis of the previously implemented Defuddle parser indicated that it is far from conforming to the DFDL specification, and also suffers from other shortcomings. We determined that it would be better to produce a new parser with equivalent features and much greater conformance to the standard. This parser is called "Daffodil", and it replaces Defuddle.

An initial implementation of Daffodil is complete, and is being evaluated to assess its conformance to the standard. We are collaborating with the OMG DFDL working group to develop an initial test suite and other conformance criteria. In addition, Daffodil is a candidate to be the first reference implementation of the DFDL standard.

7. 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.

8. References

1. Apache Software Foundation. *Welcome to JaxMe 2*. 2005, http://ws.apache.org/jaxme/.
2. 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
3. École Polytechnique Fédérale de Lausanne. *The Scala Programming Language*. 2010, http://www.scala-lang.org/.
4. Harold, Elliotte Rusty, *Processing XML with Java(TM): A Guide to SAX, DOM, JDOM, JAXP, and TrAX*, Upper Saddle River, NJ Addison-Wesley, 2002.
5. McGrath, Robert E., *Semantic Extensions to Defuddle: Inserting GRDDL into XML*. NCSA, 2008. http://cet.ncsa.uiuc.edu/CEResources/defuddle-grddl.pdf
6. 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.
7. 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

8. 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
9. Odersky, Martin, Lex Spoon, and Bill Venners, *Programming in Scala*, Mountain View, Artima Developer, 2008.
10. Rodriguez, Alejandro, *Daffodil*. Technical Note, 2010.
11. Rodriguez, Alejandro and Robert E. McGrath, *Some Notes of comparison between DFDL and Defuddle*. NCSA, 2010.
12. 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.
13. 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.