Binary OWL

Matthew Horridge, Timothy Redmond, Tania Tudorache, and Mark Musen*

Stanford Biomedical Informatics Research Group
Stanford University, California, USA

Abstract. This paper presents a binary format for both storing OWL ontologies and describing changes in OWL ontologies. The format is designed to be a fast to parse and serialise format. It is intended as a low level storage and transmission mechanism rather than an end user exchange syntax. Software to parse and serialise binary OWL has been implemented in the form of OWL API parsers and renderers. Some initial experiments seem to indicate that a Binary OWL ontology document can be parsed roughly an order of magnitude faster than the corresponding RDF/XML document.

1 Introduction

The normative exchange syntax for OWL 2 documents is RDF/XML [1] and OWL 2 compliant tools such as editors like Protégé [7], the NeOn toolkit [4] and TopBraid Composer [14], *must* provide support for parsing and serialising ontologies in this format. Even though the RDF/XML parsers and serialisers in mature APIs such as the OWL API [5] are highly optimised, it is still the case that for large ontologies, parsing and serialising them from and into RDF/XML can consume large amounts of memory (several gigabytes) and require significant amounts of CPU time (tens of seconds or minutes). For some applications such as desktop editors, while not desirable, this is not a fundamental problem. In these single-user situations users have come to expect to have to wait whilst they load or save large documents. However, for other applications, in particular web-apps such as WebProtégé [15], which may have many concurrent users and may require frequent on demand parsing and serialisation of many ontologies, the cumulative effect of these space and time requirements is unacceptable. It should be noted that these problems are not specific to RDF/XML. Indeed, the problems can arise with any RDF based syntax such as Turtle [12] or the seemingly simple N-Triples syntax. They can also arise for one way or another with non-RDF graph based formats such and OWL/XML [9], the Manchester Syntax [6] or the Functional syntax [10]. Ultimately, there is a need for a format that is geared towards high performance parsing and serialisation in terms of time and memory. This paper presents a binary ontology document format called *Binary OWL* as a possible solution to this need. Ultimately, Binary OWL is not intended to be

* This work was supported by Grant GM103316 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

used as an exchange syntax between tools (although with support in commonly used APIs it could be), rather it is designed to enable applications to support specific self-contained functionality in a performant way.

2 Preliminaries

OWL 2 OWL 2 is the latest standard in ontology languages from the W3C. For the purposes of this paper an OWL 2 ontology is a set of annotations, a set of axioms, and a set of imports declarations that can be optionally named with an Internationalised Resource Identifier (IRI) and a version IRI. Axioms are statements which specify how entities (classes, properties and individuals) or complex classes in the domain of interest are related to each other. For a full description see the OWL 2 Structural Specification and Functional-Style Syntax specification [10]. A change to an OWL 2 ontology is either an ontology annotation addition or removal, an axiom addition or removal, or an imports declaration addition or removal.

RDF Graph Based Syntaxes (RDF/XML) As pointed out above, the normative exchange syntax for OWL 2 ontologies is RDF/XML. As the name suggests, RDF/XML is an (XML) eXtensible Markup Language based format for storing RDF graphs. Since OWL 2 ontologies can be mapped into RDF graphs they can be stored in RDF/XML. To store an OWL 2 ontology in RDF/XML it is first necessary to translate it to an RDF Graph using the transformation to triples defined in Table 1 and 2 of [11] and then transform this to a stream of characters representing a valid XML document in accordance with the serialisation specification defined in [1]. The reverse process of parsing an RDF/XML document into an OWL 2 ontology is slightly more involved and is accomplished by first parsing the stream of characters into XML, parsing this into RDF triples (in accordance with Tables 3 to 18 in Section 3 of [11]) and then parsing or lifting these triples into OWL 2 ontology objects. In the context of this paper, and from the point of view of parser implementors, this syntax is difficult to parse into high level OWL 2 objects in an easy manner. At this point it should these problems are not specific to RDF/XML, but all other RDF graph based syntaxes for example Turtle, N-Triples and newer syntaxes such as JSON-LD [13]. The main cause for these difficulties is that many OWL 2 constructs, such as complex class expressions, which are single objects in the OWL world, must be represented using multiple triples. To “lift” the triples that represent a complex OWL object into that OWL object it is necessary to have all of the triples available (either in memory or in some persistent triple store) so that they can be randomly accessed and queried. In the general case, it is *not* possible to parse RDF/XML (or any RDF graph based representation of an OWL ontology) in a *streaming mode*—the triple based representation has to be fully loaded into main memory or be readily available for querying in order for it to be parsed into OWL. This has an impact on both parsing time and on the amount of memory required for parsing.

Non-RDF Graph Based Syntaxes Besides RDF/XML and the other flavours of RDF graph based syntaxes there exist several other syntaxes which can be used persist OWL ontology documents. OWL/XML is straightforward XML based syntax which can be queried and manipulated with off-the-shelf XML tools. While relatively simple, OWL/XML is extremely verbose—even more verbose than RDF/XML. This means that parsing an OWL/XML version of an ontology document can take as long, or usually longer, than parsing an RDF/XML equivalent. Other syntaxes include the Manchester Syntax, which is designed for use in editing tools such as Protégé, and the OWL 2 Functional-Style Syntax, which is used to specify the structure of objects in OWL 2 ontologies in a concrete way. All of these are textual based syntaxes which can be edited in a regular text editor.

Binary file formats Binary files are computer files that are not plain text based (files whose bytes aren't aligned to human readable characters). In recent years there has been a shift away from binary file formats to text based formats, in particular markup languages such as XML. One of the main reasons for this shift is the ease in which text based formats, such as XML, can easily be shared between 3rd party tools and can also be parsed and processed on disparate platforms (providing a common standard encoding of characters is used such as UTF-8). Another benefit of textual based formats is that they can be opened, inspected and tinkered with using a simple text editor—a comfort blanket for many people, including power users and developers. When a binary file format is opened in a text editor, the 8 bit blocks of bytes are interpreted as characters, leading to an unreadable mess. Because of this, unlike a file that is based on a textual format, it is difficult to repair a damaged or corrupt binary file by hand. This means that standards and versioning are particularly important when it comes to binary file formats—it is non-trivial to reverse engineer a binary file format. Despite good motivations for and the widespread use of text based file formats such as XML, binary formats have an advantage in terms of performance. First, binary files tend to be more compact than text based alternatives. There are typically fewer bytes to read in a binary file than the text based equivalent. Second, binary files are typically geared towards being fast to parse. In the extreme case a binary file does not need to be parsed if it is the exact copy of an in memory based representation of an object or set of objects—the bytes in the file simply need to be blitted into main memory to load the objects stored in the file. Where this kind of blitting isn't possible, binary formats can be optimised for fast parsing since they can store information in an order that is preferred for parsing rather than in an order that is more palatable to a human reader.

Binary RDF There has recently been an effort to introduce a binary format for RDF graphs [3]. The format is known as (HDT) which stands for Header, Dictionary and Triples and is essentially based on indexing IRIs and compacting the representation of a set of triples. Although using this format to store an ontology might lead to performance gains for parsing the RDF graph representation of the ontology, once the triples have been loaded into memory they would still require lifting to the level of complex OWL objects.

Binary Compatibility Cross Platform Issues One major issue for those working with binary file formats is that of byte ordering, usually known as *Endianness*. There are two flavours: *Big Endianness*, where the most signficant byte comes first, and *Little Endianness*, where the least significant by comes first. For more information and an example, see the Wikipedia article on Endianness `http://en.wikipedia.org/wiki/Endianness`. Different platforms can use different byte orderings and an important consideration in designing a binary file format is to choose one type and stick to it—a file that uses a Big Endian encoding cannot be read by a parser that expects a Little Endian encoding.

3 Binary OWL — Main Design Ideas

In what follows Binary OWL is described. It should be noted that this paper is not meant to serve as a precise specification of the syntax rather it just presents some of the salient ideas behind the syntax.

A Binary OWL Document is a binary file with a Big Endian encoding. Binary OWL uses the concept "chunks" which are blocks of binary data of a specific type. Each chunk begins with 4 bytes marking the length of the chunk, followed by 4 bytes marking its type followed by the chunk data which is of the specified length in bytes. The concept of chunking was inspired by the use of chunks in the PNG specification [2] and is a useful concept because it allows parsers to skip over chunks that they do not understand or ones that they are not interested in. Chunking also helps to make the format extensible. If new kinds of data need to be stored new chunk types can be added.

Document Header Each binary ontology document begins with a header which contains the usual information such as a *magic number* identifying that the contained document is Binary OWL and version information for the format. The header also contains a metadata chunk that can be used to store arbitrary metadata that should not necessarily be contained in the actual ontology stored in the document (such as generator, creation time etc.). The metadata chunk is akin to comments in an XML file but slightly more expressive in that property value pairs (of various types) can be stored. Some basic information (a subset of what is usually considered to be "ontology header" information) follows the document header. This ontology header contains the IRI and version IRI (if present) of the ontology contained within the document along with a possibly empty list of OWL imports declarations.

IRI Lookup Table A key component of Binary OWL documents are IRI lookup tables. Lookup tables are used to index IRIs contained in the signature of OWL ontology objects. These indices are then used in the representation of OWL objects throughout the Binary OWL ontology document. The main advantage of IRI lookup tables is that (1) they help to make the whole ontology document much more compact—replacing all occurrences of a commonly occurring IRI string, which could be tens of characters long, saves a lot of bytes. This compacting effect helps makes the file smaller and faster to read. (2) The IRI table provides an cheap interning mechanism for objects representing IRIs when the ontology is parsed—only one object is used for a given IRI and it is cheap to access these objects because they are pointed to by indices rather than being stored within some kind of map. The net effect is that fewer objects need to be created in memory which reduces parsing time and memory footprint. In every Binary OWL document there is a main IRI lookup table, which follows the document header, for the main document data block.

Document Data Following the IRI table comes the main document data block which essentially specifies a set of ontology annotations and then a set of axioms for the ontology described in the document. The next two sections below describe how components of this data block are structured. In the current version of Binary OWL a simple encoding mechanism is used which simply lists annotations and then axioms in their binary encoding. In future versions of Binary OWL one could imagine some other encoding scheme which could take advantage of the shared/repeating structure of axioms and their sub-components to produce smaller files, however, as with all compression algorithms there are some space/time tradeoffs and more investigation would be needed before setting on a preferred encoding.

Representation of OWL Objects The various kinds of objects, such as axioms, class expressions, data ranges, entities and literals that are defined in the OWL 2 Structural Specification and can be used in various places within an OWL ontology document are encoded in what is in essence a compact version of the Functional-Style syntax. Each type of object is assigned a 1 byte type marker which is written out to mark the start of the object. Next, the various sub-objects, in the order that they appear in the functional-style syntax, are written out. For example, consider SubClassOf(:A ObjectSomeValuesFrom(:R :B)). First the marker byte for SubClassOf (which has a decimal value of 36) is written out. Next, the class :A is written out by writing the marker value corresponding to class names (which has a decimal value of 4) followed by a variable length integer that is the index for :A in the IRI table. The same is repeated for the sub-object ObjectSomeValuesFrom(:R :B) and its sub-objects :R and :B.

Representation of Lists and Sets Many OWL 2 constructs consist of sets or lists of objects. For example, component of an ObjectIntersectionOf is a set of class expressions. Additionally, the main top level components of an ontology are sets (of axioms and annotations). Given a collection (set or list) of objects of size n, the Binary OWL format stores the collection using a variable length int (1 - 4 bytes) to store n followed by the serialisation of the n objects that are contained in the collection. It was decided to use a variable length int, rather than say a 4-byte int, because the use of collections can be numerous in any given ontology. In large ontologies, such as SNOMED-CT there can be huge numbers of small collections and storing the size of each collection as a 4-byte int can significantly blow up the size of the whole ontology serialisation. A further benefit of this scheme is that it enables precise advanced memory allocation during parsing. When adding items to collections such as sets, lists and maps, programs written in Java can obtain a substantial runtime performance boost by allocating enough memory to hold the complete final collection. This is especially true of HashSets which require additional memory to be allocated and moreover need to be rehashed (an expensive operation) when an item is added that requires a capacity increase of the HashSet. Thus, allocating precisely enough memory (not too little and equally important not too much) to hold all axioms and hold the numerous other sets of objects that are required in an ontology can lead to faster parsing and can decrease the memory foot print required to hold the parsed ontology in memory.

Encoding of Strings and Literals Many ontologies contain literals as the values for annotations. In Binary OWL most types of literals are represented as arrays of UTF-8 encoded characters followed by an index to the IRI representing the datatype. In terms of encoding strings, a similar mechanism to the storage of collections are used where the length of the array is encoded followed by an array of the specified length in bytes. For literals that are typed as rdf:PlainLiteral, xsd:String and xsd:Boolean more compact encodings are used that omit the datatype pointer and in some cases (e.g. xsd:Boolean) require less bytes that a raw string representation would require.

Table 1: A High Level Overview of a Binary OWL Document

Section	Field	Number of bytes	Comment
Preamble	Magic Number	4	value=BO2O
	Version Number	4	
	Metadata chunk	variable (chunk)	
Ontology Header	OntologyID	variable	
	Import declarations	variable	
Index Table	IRI Table		
Document Data	Annotations	variable (chunk)	
	Axiom Table	variable (chunk)	
	Change List	variable	unclosed list

4 Incremental Update

As well as a need for performant parsing and serialisation capabilities, applications typically have robustness requirements which can dictate the choice of document storage technologies. For example, if an ontology editing application crashes, or the machine that it is hosted on needs to be restarted, there is potential for data loss—any changes that have been made since the last save operation will be lost. This problem can be mitigated to some extent by persisting ontology changes to disk *as they happen*. However, the normative and other main exchange syntaxes such as OWL/XML are XML based meaning that it is not possible to *append* changes to existing documents. The upshot of this is that for small document changes, such as adding an axiom or annotation to an ontology, the *whole* ontology document needs to be recreated and rewritten out for that ontology. The traditional approach to this has been to use database back end stores, for example [8]. However, these stores are heavy weight solutions and tend to suffer from performance problems caused by frequent reads and writes. Having a format which can accommodate ontology changes as appendages would be beneficial in this situation. Binary OWL therefore makes it possible to append a lists of changes to an existing document. Each list of changes is represented as a chunk, with its own metadata chunk, IRI table (scoped to IRIs appearing in the list of changes), and finally a list of the changes themselves. The following changes are supported:

- Add axiom
- Remove axiom
- Add ontology annotation
- Remove ontology annotation
- Add imports declaration
- Remove imports declaration
- Set ontology Id (set the ontology IRI and possibly the version IRI).

The above list of changes parallels the basic change types in the OWL API. At this point, it is worth mentioning that Binary OWL is strongly typed like the functional syntax, which means that objects can be parsed in isolation without reference to declarations the rest of the ontology or imported ontologies. This means it is possible to modify the imports closure in the appended changes without any side effects being caused by type declarations being added or removed from the imports closure.

5 Experiments

In order to determine the differences between parsing a file stored in the normative RDF/XML exchange syntax and Binary OWL, and to look for further areas of optimisation, we implemented an OWL API parser and renderer for Binary OWL and carried out some small scale casual experiments. It should be noted that we decided that a comparison should be made with parsing RDF/XML rather than, say OWL/XML, for two reasons: (1) RDF/XML is the normative (and most widely used) OWL exchange syntax that all OWL tools must support, and (2) some pilot experiments using the OWL API indicated that for a given ontology its RDF/XML parser provides faster parsing performance than its

OWL/XML parser. Therefore, the results which follow also hold for OWL/XML in the sense that performance gains provided by Binary OWL over RDF/XML will also be performance gains for Binary OWL over OWL/XML. The software is written in Java and uses the `java.io` classes `DataInput` and `DataOutput` for reading and writing primitive data such as bytes, ints and strings in a cross-platform way by using a big endian encoding.

Several well-known large ontologies ranging from 886,578 axioms in size to 6,062,769 axioms in size, were used for the experiments. The ontologies are shown in Table 2, which shows the number of axioms, the number of logical axioms, the number of annotation axioms, the file size of the RDF/XML file in Megabytes and the file size of the Binary OWL file in Megabytes.

Method To ensure comparable results over the corpus and eliminate network delays each ontology and its imports closure was first loaded with the OWL API (version 3.4.3), processed so that the imports closure was merged into one ontology, then saved into an RDF/XML file and also a binary OWL File. Next, for each ontology its RDF/XML file and its Binary OWL file were parsed by the OWL API RDF/XML parser and the Binary OWL parser implemented as part of this work. The CPU time taken to parse each file *alone*, ignoring the time taken to index objects into the OWL API data structures, was recorded and averaged over 10 rounds. The results are shown in Table 3.

Table 2: Ontologies used in Experiements

Ontology	**Axs**	**Log. Axs**	**Ann. Axs**	**RDF MB**	**B.OWL MB**
SNOMED	886,578	295,492	591,086	254	36
Mesh	975,855	403,210	572,645	105	25
NCI Thesaurus	1,212,839	130,945	1,081,894	213	59
Bio Models	1,905,822	660,188	1,245,634	275	73
NCBI Taxon	6,062,769	847,755	5,215,014	814	179

Table 3: Results of Parsing the Ontologies in RDF/XML and Binary OWL (parsing times are CPU times and are shown in seconds)

Ontology	**RDF/XML CPU Time/(s)**	**Binary OWL CPU Time/(s)**	**Speed Up**
SNOMED	18.69	1.55	12.0
Mesh	7.84	0.86	9.15
NCI Thesaurus	13.60	1.71	7.96
Bio Models	20.54	2.52	8.15
NCBI Taxon	59.10	6.56	9.01

Analysis As can be seen from Table 2, the size of a Binary OWL file is roughly 4 times smaller than the corresponding RDF/XML file. The main compression comes from using IRI indexes and from compressing away the language vocabulary that is used in RDF/XML (and other serialisations such as the Functional-Style Syntax). Even though the RDF/XML parser in the OWL API is highly tuned (even very large ontologies such as the NCBI Taxon ontology, which contains over 6 million axioms, can be parsed in approximately 1 minute of CPU time) parsing the binary OWL version of an ontology document is on average an order of magnitude (between 8 and 12 times) faster than parsing the RDF/XML version of the document. In some cases the time difference, e.g. 6.56 seconds versus 60 seconds for the NCBI Taxon ontology, could have a large impact on applications that require frequent loading of large numbers of ontologies (or indeed very frequent loading of lots of smaller ontologies).

6 Conclusions

This paper has introduced the idea of a binary format for OWL. The main motivation for this work was to produce a file format geared towards the fast parsing and serialization of OWL ontologies. The format is intended for internal application use rather than being yet another exchange syntax or human consumption format. Although simple, with further room for optimisation, the format looks promising and is roughly an order of magnitude quicker to parse than RDF/XML.

Finally, although the Binary OWL spec is relatively stable, some small details are still being finalised. A release of some libraries for working with Binary OWL is therefore part of future work. Developers who are interesting in trying out the format in its current form should contact the authors.

Acknowledgements This work was supported by Grant GM103316 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

References

1. Dave Beckett. RDF/XML Syntax Specification (Revised). Technical report, World Wide Web Consortium, February 2004.
2. David Duce. Portable network graphics (PNG) specification (second edition). Technical report, World Wide Web Consortium, 2003.
3. Javier D. Fernández. Binary rdf for scalable publishing, exchanging and consumption in the web of data. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012*, pages 133–138, 2012.
4. Peter Haase, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d'Aquin, and Enrico Motta. The neon ontology engineering toolkit. In Jeff Korn, editor, *WWW 2008 Developers Track*, April 2008.
5. Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, February 2011.

6. Matthew Horridge and Peter F. Patel-Schneider. Manchester OWL Syntax for OWL 1.1. In *OWL: Experiences and Directions (OWLED)*, 2008.
7. Matthew Horridge, Dmitry Tsarkov, and Timothy Redmond. Supporting early adoption of OWL 1.1 with Protégé-OWL and FaCT++. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *OWL: Experiences and Directions (OWLED)*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, November 2006.
8. Joachim Kleb Jörg Henss and Stephan Grimm. A database backend for OWL. In Rinke Hoeksta and Peter F. Patel-Schneider, editors, *OWL: Experiences and Directions (OWLED 2009)*, CEUR Workshop Proceedings. CEUR-WS.org, October 2009.
9. Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language XML serialization. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.
10. Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language structural specification and functional style syntax. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.
11. Peter F. Patel-Schneider and Boris Motik. OWL 2 web ontology language mapping to RDF graphs (second edition). Technical report, World Wide Web Consortium, December 2012.
12. Eric Prud'hommeaux, Tim Berners-Lee, Dave Beckett, and Gavin Carothers. Turtle terse RDF triple language W3C candidate recommendation. Technical report, World Wide Web Consortium, February 2013.
13. Manu Sporny, Gregg Kellogg, and Markus Lanthaler. JSON-LD 1.0 a JSON based serialization for linked data. W3c editor's draft, World Wide Web Consortium, March 2013.
14. TopQuadrant. Topquadrant: Products: TopBraid Composer. `http://www.topquadrant.com/products/TB_Composer.html`, October 2009.
15. Tania Tudorache, Csongor Nyulas, Natalya Fridman Noy, and Mark A. Musen. WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web*, 4(1):89–99, 2013.