`drys`: A Version Control System for Rapid Iteration of Serialization Protocols

Extended Abstract

Jason A. Wilkins
Texas A&M University
College Station, TX
jwilkins@cse.tamu.edu

Jaakko Järvi
Texas A&M University
College Station, TX
jarvi@cse.tamu.edu

Abstract

Tagged binary formats are fast to load but potentially fragile to maintain as a software system evolves. This paper presents a tool for generating the code not just for serializing binary file formats, but also for upgrading old versions to newer ones, enabling rapid development of new versions.

1 Introduction

Binary files are very efficient compared to text files because they can be directly mapped into memory. Text files must be parsed. However, to keep old versions of binary files from becoming obsolete, the code base must contain a copy of the code for loading the old format, and optionally, code for converting it to the new format. The maintainability of this approach decreases as the number of versions increases. Text files admit a more incremental approach where new features can be added by making the syntax of the new features a superset of the old formats. Tagged binary formats reduce the impact of changes to smaller regions by tagging regions with a format label, and allowing a loader to skip any unknown regions of the file. However, this only delays the problem by making it smaller (basically a smaller constant factor instead of an asymptotic reduction in maintainability).

This work applies the concepts of version control *not* to the code that implements file formats, but to the resulting file formats themselves. Maintenance is simplified because the developer only has to work with code that represents the current *HEAD* revision of the data format. All previous versions are kept in a repository and are used to generate code that can load previous versions and upgrade them to the *HEAD* revision.

Ambiguous changes to the file format (considering the history of the file) are treated as errors. For example, changing the name of a field is ambiguous because it cannot be determined if the edit is meant to change the name or delete an old field and add a new field. However, it is not an error to achieve this incrementally. To rename the developer could commit an annotation that the field name has been changed, and then remove the annotation and commit again. To show that the field is unrelated but occupies the same location as the previous field, the developer can add the new field and commit, then remove the old field and commit. Alternatively he could annotate that the field as not having been renamed. The trick here is that the annotations have to be used to resolve ambiguous edits, but they do not have to live in the file forever, since the repository maintains

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

them. This allows the programmer to only see the file format as it exists in the *HEAD* but still have the ability to confidently deal with ambiguities as they arise.

File formats are typically thought of as being linearly related. However, this system would only require that there be a directed path from a previous file format to the *HEAD* in order to convert. This allows for files created by a feature branch of an application to eventually be loaded by the master branch after a merge. Conversion is always possible because all data has a constructor which can potentially rely on previous versions of the data (or a default if there was no previous version of a field). Dataflow analysis through previous versions of the data constructors allows for even the oldest version of a file format to be updated to the *HEAD*.

Incidental (or even accidental) file formats created by a programmer during development no longer mean that the files created with those accidental file versions cannot be loaded. The system can keep track of all incrementally working file formats as *weak commits* that can be integrated into development builds, but left out of release builds.

The scenario this system is most useful in is where an application has a large and complex set of data structures that it maintains internally which need to be serialized efficiently, but for which it would be onerous to strictly version control due to distributed development. This system would allow for rapid iteration of binary file formats with confidence that any files saved, even by development versions, could be loaded by any other future version.

2 Previous Work

2.1 Blender

Blender implements a system called *DNA* which attempts to solve the problem presented here (rapid iteration of a binary format in distributed development), but which has several shortcomings. It has a tool called `makesdna` which reads `C` header files and generates serialization code for `C` data structures. The format of `blend` files is such that old versions can be upgraded to new versions by comparing field offsets and running the file through a function that has been hand written to set new default values and convert old values. If the file version matches the executable version then the file can be loaded very quickly compared to similar applications. (Users have jokingly referred to how they can go get coffee when opening Maya or 3D Studio Max.)

2.2 Domain-Specific Languages for Binary Formats

PADS [FG05] is a declarative domain-specific language for specifying the layout and semantics of ad hoc data sources. It is meant to be readable enough to serve as both the documentation and specification of the formats.

Google Protocol Buffers and Cap'n Proto serve similar purposes.

2.3 Standard Formats

Standard or de facto standard binary formats exist for the interchange of a myriad different kinds of data and should be used wherever it is possible to do so.

2.4 XML Encoded Descriptions

The scientific computing community has addressed the problem of the need to share data while dealing with incompatible binary file formats. The Data Format Description Language [WWC03] (DFDL, pronounced daffodil), encodes binary formats in XML. It separates the structure and semantics into separate descriptions. The goal with this kind of format is interoperability between different sources of data, where `drys`is attempting to solve the problem of incompatibilities in binary formats in what may be single project over time. Using DFDL, the Defuddle [TSSM06] parsing engine converts between various scientific binary formats.

The goal of format description called XCL is to use binary format description languages to preserve the meaning of file formats apart. The meaning of a binary file is often only embodied as a de facto specification by the programs that reads them (which might only be runnable through emulation) [BRH+08]. `drys` is also concerned with preservation, but in a different way, as it keeps the developmental history of a file format, thus preventing a program from accidentally not being able to read data produced by an older version of the program.

The goal of XTDM [MZF+05] is to present an abstract data format description that bridges binary files, data bases, and spreadsheets. It accomplishes this by separating the abstract view of the data from the representation description and then defining a procedural mapping between the two.

The XML encoded descriptions are parallel developments to `drys`, since the contribution is history sensitive compilation, as there is nothing really special about `drys`'s binary description format (except that it is not encoded in XML due to the way it is meant to be used by developers).

2.5 Binary XML

Binary XML for Scientific Applications [CDLS05] is a binary encoding of XML (not a description of a binary format encoded in XML). The purpose of a XML binary encoding is to speed the parsing of structured data by removing much of what makes processing text slow, while creating a flexible format that puts scientific data on the same footing as other data that can be accessed through web interfaces.

There is also a W3 specification [MJ99] for binary XML.

2.6 Self-Describing Formats

NetCDF [RD90] is a self-describing format, which means that it contains the information needed to decode the format. Another self-describing format is NSCA HDF (Hierarchical Data Format) [FMY99] is a set of tools used to manage scientific data. These are still yet other formats for scientific data interchange, but they do not appear to be based on XML.

3 The `drys` System

`drys` is a software maintenance tool that generates serialization code with the additional contribution of also generating the code required to upgrade old messages and files to newer versions. The basic functionality is similar to Google's Protocol Buffers or Cap'n Proto, which are software libraries designed to generate binary format file/network wire protocols for efficient information interchange.

`drys` focuses on the problem of rapidly evolving versions of the serialization protocol, including being able to handle decentralized version changes and temporary intermediate versions. Rapid development of new versions can occur in an application that frequently adds, removes, and modifies features. This is complicated by the fact that new versions of the program are almost always required to exchange data with previous versions of the application smoothly (backwards compatibility).

3.1 Version Hell

Code written to insure backwards compatibility code checks for old versions of the serialization protocol and adds new fields with default values, ignores obsolete fields, and rearranges or updates modified fields based on old values. Updating an old message or file written by an old version of the serializer is like replaying the history of the protocol. The simplest approach is to upgrade each outdated field in turn to correspond to each newer version until the file/message is updated. The code may resemble the following example, but go on for thousands of lines.

```
uint8_t read_wind_seed()
{
    uint16_t wind_seed12;
    uint8_t  wind_seed23;

    // In version 12 we added a customizable seed for wind turbulence
    if (version < 12)
        wind_seed12 = rand16(time()); // this line duplicates code from a long gone constructor
    // In version 16 we added another seed so we renamed 'seed' to 'wind_seed'
    else if (version < 16)
        wind_seed12 = read16(SEED_TAG); // reads a the field using an earlier name
    else if (version < 23)
        wind_seed12 = read16(WIND_SEED_TAG); // the seed used to have 16 bits
    else
        wind_seed23 = read8(WIND_SEED_TAG); // latest version (23+)

    ...
```

```
    // In version 23 we decided to limit seeds to 8-bits, and
    // generate new seeds by xor'ing the old bytes together.
    if (version < 23)
        wind_seed23 = (wind_seed12 >> 8) ^ (wind_seed12 & 8);

    ...

    return wind_seed23;
}
```

The open source application Blender has well over 10,000 lines written in this way, all by hand, which covers its 20 year history.

Writing backwards compatibility code by hand is a tedious and error prone activity and the resulting code can become long, complex, fragile, and inefficient.

4 Comparison to other Version Control Systems

Distributed versioning of a serialization protocol can occur when a project is branched to develop new features in isolation. This creates an alternative history which has to be merged back into the common version.

The simplest approach to merging branching versions is to "quash" the history of the branch, which is to treat it as a single new version with no intermediate history. The disadvantage of this is that any data created by intermediate versions of the branch may have to be abandoned because the history is forgotten.

Concepts like history, branch, and quashing come from software version control systems (VCS), but the meaning of version is somewhat different. VCS is typically only aware of the textual differences, but `drys` also has to keep track of semantic differences.

Textual difference commonly represented by describing the smallest number of edits required to modify the text (although tools like diff and patch will do this line by line, not character by character). The shortest edit distance between protocol versions can be ambiguous. In textual editing, renaming a field can be represented as deleting the old line and adding a new one and it does not matter if the intent was to update the field, but with protocol editing this is the difference between dropping old data while adding new data, or simply reading old data under a new name.

It is not possible to automatically extract new versions by examining the textual (shortest edit distance) history alone. `drys` requires disambiguation, which can be handled interactively or by annotating the code. For similar reasons, `drys` needs to examine the code every time it is compiled, not just when it is committed to the VCS, because uncommitted versions of the protocol may serialize data that needs to be used for testing. Updating and later removing code used only to upgrade a fleeting version of the protocol would be especially tedious. These intermediate developmental versions are marked so they can be quashed once testing is completed.

Acknowledgments

The work was supported in part by NSF grant CCF-1320092.

References

[BRH+08] Christoph Becker, Andreas Rauber, Volker Heydegger, Jan Schnasse, and Manfred Thaller. A generic XML language for characterising objects to support digital preservation. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 402–406. ACM, 2008.

[CDLS05] Kenneth Chiu, Tharaka Devadithya, Wei Lu, and Aleksander Slominski. A binary XML for scientific applications. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. IEEE, 2005.

[FG05] Kathleen Fisher and Robert Gruber. PADS: a domain-specific language for processing ad hoc data. In *ACM Sigplan Notices*, volume 40, pages 295–304. ACM, 2005.

[FMY99] Mike Folk, Robert E McGrath, and Nancy Yeager. HDF: an update and future directions. In *Geoscience and Remote Sensing Symposium, 1999. IGARSS'99 Proceedings. IEEE 1999 International*, volume 1, pages 273–275. IEEE, 1999.

[MJ99] Bruce Martin and Bashar Jano. WAP binary XML content format. W3C note. *World Wide Web Consortium (June). Cambridge, MA*, 64, 1999.

[MZF+05] Luc Moreau, Yong Zhao, Ian Foster, Jens Voeckler, and Michael Wilde. XDTM: The XML data type and mapping for specifying datasets. In *Advances in Grid Computing-EGC 2005*, pages 495–505. Springer, 2005.

[RD90] Russ Rew and Glenn Davis. NetCDF: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

[TSSM06] Tara D Talbott, 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 *Provenance and Annotation of Data*, pages 73–81. Springer, 2006.

[WWC03] Martin Westhead, Ted Wen, and Robert Carroll. Describing data on the grid. In *Grid Computing, 2003. Proceedings. Fourth International Workshop on*, pages 134–140. IEEE, 2003.