DATASET VERSIONING THROUGH CHANGELOG ANNOTATION

By

Benno Lee

Prepared for:

Peter Fox, Thesis Advisor

Jim Hendler, Advisor

Deborah MacGuiness, Member

Beth Plale, Member

Rensselaer Polytechnic Institute
Troy, New York

November 2016
(For Graduation December 2017)

CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

ACKNOWLEDGMENT vi

ABSTRACT vii

1. INTRODUCTION 1

    1.1 Types of Change 2

    1.2 Provenance and Provenance Distance 3

        1.2.1 Provenance Distance 3

    1.3 Changelogs 4

    1.4 RDFa 5

2. PREVIOUS WORK 6

3. CONCEPTUAL MODEL 7

    3.1 ADDITION 7

    3.2 INVALIDATION 8

    3.3 MODIFICATION 8

    3.4 MULTIPLE LINKED VERSIONS 9

4. VERSIONING SPREADSHEETS 10

    4.1 Provenance Analysis 10

    4.2 Versioning Comparison 11

5. DATABASE VERSIONING 13

6. ONTOLOGY VERSIONING 14

7. CONCLUSION 15

REFERENCES 16

LIST OF TABLES

LIST OF FIGURES

3.1 Model of the relationships between Versions 1 and 2 when adding an Attribute 2 to Version 2 as a result of Change A 8

3.2 Model of the relationships between Versions 1 and 2 when invalidating Attribute 1 from Version 1 as a result of Change I 9

3.3 Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2 9

4.1 Provenance graph for the entry CAM001 entry of the Noble Gas Database. Other than the labels, the structure of each of the data objects is very much the same. 11

ACKNOWLEDGMENT

This is a sentence to take up space and look like text. This is a sentence to take up space and look like text. This is a sentence to take up space and look like text.

This is a sentence to take up space and look like text. This is a sentence to take up space and look like text. This is a sentence to take up space and look like text.

This is a sentence to take up space and look like text. This is a sentence to take up space and look like text. This is a sentence to take up space and look like text.

This is a sentence to take up space and look like text. This is a sentence to take up space and look like text. This is a sentence to take up space and look like text.

This is a sentence to take up space and look like text. This is a sentence to take up space and look like text. This is a sentence to take up space and look like text.

This is a sentence to take up space and look like text. This is a sentence to take up space and look like text. This is a sentence to take up space and look like text.

This is a sentence to take up space and look like text. This is a sentence to take up space and look like text. This is a sentence to take up space and look like text.

ABSTRACT

This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text.

This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text.

This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text.

This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text. This is a sentence used to take up space and look like text.

CHAPTER 1
INTRODUCTION

Software development followed a stiff production cycle prior to the early 2000s. However, as technology developed, software development required a system to adapt and change to evolving conditions leading to the Agile Manifesto. Likewise, data collected by researchers grew at an astounding rate with new technology. NASA's Atmospheric Science Data Center reported a growth from hosting around five million files to twenty million files between 2001 and 2004[1]. Many datasets have required reprocessings of their data, either to improve data quality or to correct for errors [CALIPSO, ARM]. Indeed, version control systems can provide contexts for modifications when combined with error detection systems, providing a more complete picture of the system's behavior [3]. (Dynamic dataset generation?)

Dataset versioning tracks and documents the changes which occur in datasets. There exists a tendancy to model dataset change using methods from software versioning [4] [5] [6]. Dot-decimal style version numbers are commonly seen in relation to new releases of software such as MATLAB or R, and current version naming schemes in data often use dot-decimal notation. However, this convention results in semantically poor version names as the decimal numbers generally function solely as counts [7]. The Ubuntu versioning scheme appears to follow a major-minor numbering pattern, but the major and minor numbers correspond to the year and month of the release, respectively. As a result, the version name holds no significance as the extent of difference between two versions. Additionally, this method breaks down since it is difficult to predict the impact changes in data will have on data consumers. Depending on how the data is processed or what subsets are used, dataset changes that the producers perceives as minor changes may have significant repercussions for the consumer. This demonstrates the structural difference between software and datasets in that a revision in data only constitutes an adjustment to the relevant

[1]Agencies and research groups have collected new data at an incredible rate. The amount of data housed by NASA quadrupled from 2001 to 2004 [1] and high energy physics labs can generate on the order of 4000 new datasets every day [2].

subsets. An alteration to a software file constitutes a change to the project as a whole. As a result, datasets provide a context which does not completely translate from software project versioning.

1.1 Types of Change

Change is the fundamental characteristic in the study of versioning. Continuing the comparison with software, the measure of change in applications is primarily functional. Data, on the other hand, has not only function but also structure. While swapping the ordering of columns in a spreadsheet may not effect the usability of the data, it could cause issues with codes that use the data. This highlights the idea that many different forms of change exist with differing degrees of intensity. In this document, change is viewed as having three general categories: scientific, technical, and lexical. The most impactful change, scientific, denotes changes that have modified the fundamental science used to generate the dataset. Changes in algorithms or sampling methods generally fall under scientific modifications as they effect the base assumptions and possible error within the dataset. Technical impacts do not change the underlying science of the data, but impose a large enough change as to warrant notice. Structure alteration and unit conversions count as technical changes since the dataset now needs to be consumed differently but remains valid for use. Lexical changes belie the transformations that can best be described as corrections. Filling in previously missing values or fix erroneous values may be lexical changes.

The exact category that a particular change falls into can be controversial. The decision to change concentration units from parts per million to milligrams per milliliter poses a Technical change for a data producer. However, for a data consumer, the change may be viewed as a Scientific change as it invalidates the methods they had previously used. This conflict in view illustrates the data consumer-producer dynamic. In general, data producers are in control of the methods of versioning, but data consumers determine the classification of a data change. Reference 19 demonstrates the viewpoint as producers tend to use versioning systems to ensure data quality of service through audits and recovery tools. Meanwhile, a consumer will analyze the historical changes and determine the impact this may

have to their data use. As a result, this means that data versioning systems must communicate a dynamic view of the changes in a system contextualized by the user of that data.

1.2 Provenance and Provenance Distance

The information that details the activities and agents involved in generating a data entity is known as provenance. In NASA remote sensing data, provenance is generally grouped into input data, scripts, and calibration constants [Barkstrom] and changes to it signals that a new versions should be propagated down the workflow. Software revision management tools such as Git and SVN also keep track of provenance information when logging new commits to a project, using branches and merges. It is, however, important to distinguish between the contributions of provenance to change tracking and the contributions of versioning. Provenance allows for the identification of change localities; it signals where changes in data have occurred. However, provenance does not elaborate on the extent of the alterations made, not should it. The priority of provenance tracking is understanding where and how a data object was generated, not it's relationship to other data products. In the field of semantic technologies, the W3C recommendation, PROV provides a data model to encode provenance information into searchable graphs. PROV expresses relationships between versions with the wasRevisionOf or alternateOf property. The properties do not allow for more elaboration as to what changes were made to transform from one version into its alternate. Versioning provides the context for a change by providing the comparison between data objects.

1.2.1 Provenance Distance

Data workflows have become complex and sizable in such a way that even subtle changes may have noticeable effects. (Refer to NASA citation of cosmetic changing resulting in changed datasets). One approach is following the provenance of the new data and identifying potential differences between how the data has become generated. Since provenance graphs have both directed edges and labeled nodes, there are a few methods of measuring similarity available (Levenshtein Distance)

beyond basic isomorphism. This similarity measure is known as the provenance distance between the two data objects. A concern that arises with this approach is that, as previously stated, provenance only provides a context for versioning. Consider an example where there are versions 1 and 2 of a data object, both compiled from a reference source. Since there are two versions, it is understood that the versions are different so the comparison extends to their reference sources. If both versions were compiled from the same source, it can only be concluded that the compilation methods were different. In the case of the NASA datasets, those methods should not have changed and no further insight has been found. If the reference sources have been changed, then it is expected that there would be a new version or data object and no new information has been found. This reasoning motivates the development of a versioning method that utilizes changelogs and RDFa to determine versioning distance.

1.3 Changelogs

Changelogs, sometimes called patch notes, are artifacts resulting from the versioning process often found in major software projects. They detail the changes made within the system and some will have explanations on the motivations behind changes. These logs provide a great source of value to developers as they can be used to give insight to the health of a software project (Changelog references). Changelogs also allow developers to link bugs and errors with their corrections in later versions (Bugzilla citation). Despite these contributions, changelogs are rarely found along with data. Possible reasons for this omission may include the size of data as it may become difficult to scalably communicate data change (Data exceeds ability to track reference) or that the ease of data acquisition and storage inclines towards simply using the latest version. As a result, new dataset versions are often accompanied by usage documents, but documentation on version transitions are omitted. This creates a impediment to making data forward and backwards compatible, costing data consumers both time and money.

Changelogs are also difficult to consume as they are often built to be read only by humans. The transition between different versions of large datasets is then left

largely up to the human user's ability to understand and process the modifications mentioned within the change log. In the absence of any sort of change notes, it is up to the consumer to be able to locate the points of differences between the versions they are working among. Therefore, providing an machine consumable changelog would accelerate and assist in navigating through dataset changes and error corrections.

1.4 RDFa

The modifications to implement a machine readable changelog should still remain in a form viewable by a human user. The subjectivity of versioning information means that domain scientists should still be able to interpret changes as they relate to their application. Resource Description Framework in Attributes (RDFa) allows web developers encode information XHTML and HTML documents using standardized ontologies. The developer would use the attributes in RDFa to augment the existing HTML tags to provide extra meaning that a web crawler can extract. The displayed text would remain unchanged signifying that if a changelog could be produced in HTML, then data producers can embed machine readable versioning information into a changelog using RDFa while still leaving it human readable as well.

CHAPTER 2
PREVIOUS WORK

PROV is a W3C recommendation that deliniates a method to express data provenance with semantic technologies. Using the model of relating activities, agents, and entities, data managers can express the origins of their datasets. However, when an entity is revised, the PROV data model can only express the relationship as a revision or that the new dataset was derived from the original. This leaves

CHAPTER 3
CONCEPTUAL MODEL

The conceptual model used within this thesis is built around the expression of three core versioning operations: addition, invalidation, and modification. These three activities can be represented by interacting with three types of concepts: versions, attributes, and changes. Versions represent the data entities being compared. These could be two different editions of a book or versions of software. It is important to understand that a version is an abstraction as it can be represented by multiple physical files. In the sections that follow, operations will only consider the interaction between two versions and will be explained later in the chapter. Versions then contain attributes representing a quantity being modified. Specifically for tabular data, attributes would correspond to an identifier that refers to particular rows or columns within the data. Attributes of the two versions are then connected by a change. This link functions as a very general concept which can be subclassed into more informative types such as unit changes, improving the expressiveness of the model beyond PROV's revisionOf concept.

3.1 ADDITION

When a change adds a new attribute to a version, it only needs to refer to version two and its corresponding attribute. The reasoning should be fairly obvious as the attribute never existed in version one, and therefore, there is nothing to refer to and no need to form a relationship between the change and version one. However, by linking the addition change to version one, we address a difficulty with comparing provenance graphs. When two data objects have identical structures, it is difficult to determine what time the objects were added to the dataset and which version they belong to. As a result, determining the compatability of the two objects becomes difficult. The change contributions to the dataset evolution appears naturally using this construction. The resulting model can be seen in Figure 3.1. Some relationships are specifically left out, such as that between Change A and Version 2, to not confuse

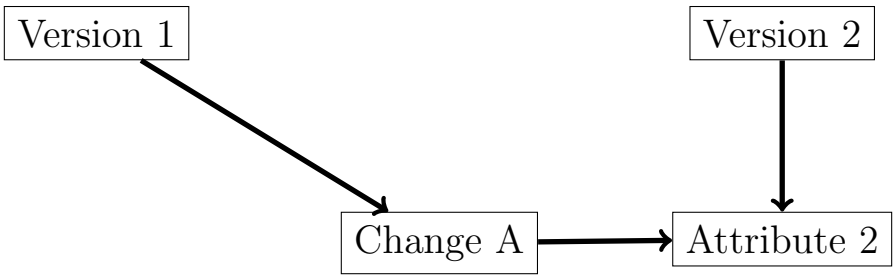

Figure 3.1: Model of the relationships between Versions 1 and 2 when adding an Attribute 2 to Version 2 as a result of Change A

identification of other types of changes. The relationship between Change A and Version 2 can still be implied from Attribute 2.

3.2 INVALIDATION

The Invalidation operation corresponds to the delete concept found in other applications. The choice of invalidation over delete results from the policy that, in versioning, data should never be deleted. In practicality, this may not be particularly feasible due to space limitations and relative validity. In either case, the change invalidates an attribute in version one, resulting in version two. Unlike the Addition operation, Invalidation forms a clear relationship between both versions, which can be seen in Figure 3.2. Notice again that since Attribute 1 no longer exists in Version 2, there is no corresponding Attribute 2 to refer to.

From Figure 3.1, we can see the confusion that could result from requiring explicit relationships between versions and changes in both the Addition and Invalidation operations. Linking Change A to Version 2 would create a duplicate connection and provides a mechanism to identify when items specifically enter or leave a version.

3.3 MODIFICATION

The final operation is Modification, and it maps a change from one attribute from version one to its corresponding attribute in version two. The particular type of change in this case is purposely left out in order to allow data producers to subclass and customize the resulting graph to properly reflect the versioning that they desire.

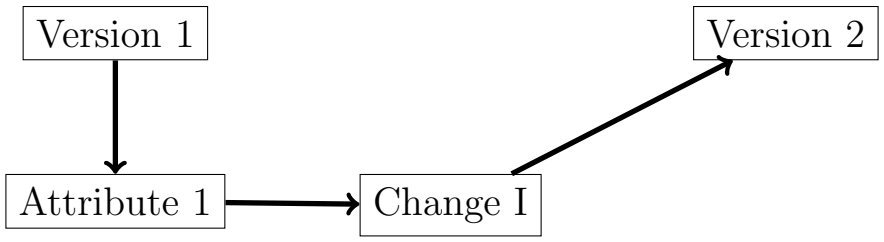

Figure 3.2: Model of the relationships between Versions 1 and 2 when invalidating Attribute 1 from Version 1 as a result of Change I

3.4 MULTIPLE LINKED VERSIONS

Using the construction outlined in the previous three sections, many changes can be compiled together into a graph in a changelog. After all additions, invalidations, and modifications have been compiled into a single graph, a complete mapping from version one to version two may be developed. The orientation of the relationships in the graph allows a flow to be created from attributes in version one to corresponding attributes in version two. Taking version two and performing the same graph construction to a version three results in not only a flow from version two to version three, but also from version one to version three. As a result, the flow can be used to construct a mapping from version one to version three or any future version.

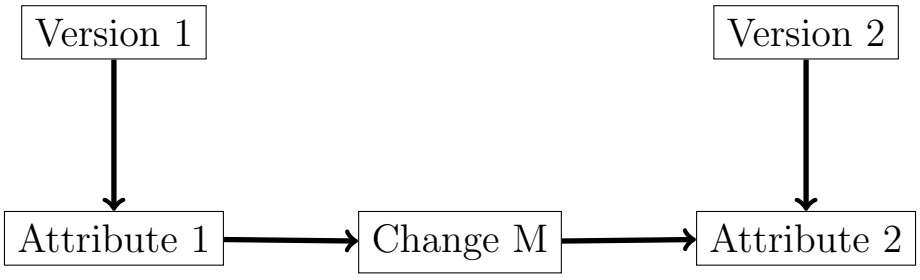

Figure 3.3: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

CHAPTER 4
VERSIONING SPREADSHEETS

Two datasets were initially studied. The "Noble gas isotopes in hydrocarbon gases, oils and related ground waters" database, initially published on June 11, 2013 and then republished a second version on March 8, 2015. The physical structure of the database changed from eight separate Excel spreadsheets to a single sheet. The model does not explicitly account for a change like this except that the identifiers used to refer to the versions or attributes involved may show that the items from version one originate from different files. The original dataset also had 195 columns. However, these were reduced to 54 columns in the second release. In addition, many new locations were surveyed and added to the second release. These are the most challenging

The Paragenetic Mode for Copper Minerals database produced two versions, one at a workshop on June 8, 2016 and another at a following workshop on August 21, 2016. These both take the form of Excel spreadsheets, which has the benefit of having strict row and column numbering. This allows unique identifiers to be used when referring to individual pieces of data and providing a level of abstraction. The structural changes made to the Copper Dataset resemble those found in the Noble Gas Dataset.

4.1 Provenance Analysis

The first approach to determining the provenance distance for the datasets began with the Noble Gas Dataset. The dataset provides a set of references from which the values were extracted and compiled into the dataset. As a result, a simple provenance mapping was constructed, using PROV, from each reference to its corresponding row in the spreadsheet. After this was done for each version of the dataset, we can generate graphs to compare objects from the two versions like the one found in Figure 4.1. We can tell from the labels that different Activities were used to compile the data entry, but the structure of the graph does not provide

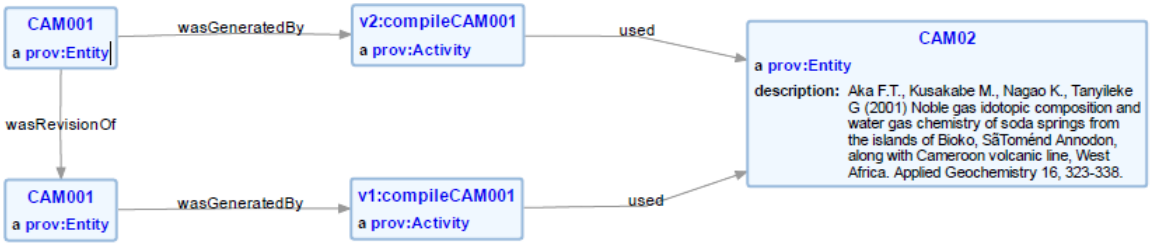

Figure 4.1: Provenance graph for the entry CAM001 entry of the Noble Gas Database. Other than the labels, the structure of each of the data objects is very much the same.

any information as to how extensive this change was for this version. Here we can see how the wasRevisionOf relationship would break down in determining the magnitude of change between the two versions. The relationship must, therefore, be expanded in order to provide the desired data necessary to make an evaluation.

4.2 Versioning Comparison

The initial challenge for both of these datasets is producing an appropriate mapping between their previous and latest versions. Through the second compilation process, many columns were removed from the dataset or moved around, and as a result, the identifiers associated with the Attribute in version two did not necessarily correspond to their identifier in version one. The column headings in the Noble Gas Dataset is stored in the spreadsheet as data and makes it difficult for a general system to automate mapping columns between the two versions since any number of rows may contain metadata information. Additionally, while the second release of the Noble Gas Dataset did have a document detailing the organization of the columns contained within the dataset, it did not have any information to map entries from the old dataset to the new dataset. This limits the ability to map and update changes to human actors. Immediately apparent is that any columns that have a mapping from version one to version two means that these columns (Attributes from the model) will only undergo Modification operations since there exists an associated Attribute in both the previous and current version. The conclusion then follows that all remaining Attributes (including both columns and rows)

belonging to version one were Invalidated and those belonging to version two were Added. Using this conclusion, we can pre-calculate the Added and Invalidated Attributes and separate any report into changes grouped by operator.

Once a mapping exists between the two versions, a comparison was performed to determine whether an Attribute of the data object changed. In this case, a simple equality operator was used to determine if anything was modified. In practice, more advanced or complicated methods can be used to determine equality. Since, all Additions and Invalidations have already been predetermined, we can output each Modification as we see them. As is common in versioning, the changes were outputted to a changelog document formatted using HTML. The idea here is that by providing the changelog in HTML, the changes can be made available online and therefore accessible by data consumers. Another benefit of providing a changelog in HTML is that the document can be additionally enriched by RDFa. The changelog document becomes no longer restricted to human consumption, but allows autonomous agents to more intelligently interact with dynamic data systems. The conceptual model detailed in Chapter 3 becomes encoded into the changelog and the graph resulting from the log can be extracted automatically.

The resulting graph provides a structure upon which a flow may be calculated. This becomes an alternative method to determine the provenance distance between two datasets with a much higher fidelity. As mentioned previously, the Change concept is meant to be sub-classed to provide more freedom to represent the particular change bridging the two versions. For example, the He Count entry for the Noble Gas Database changed its units from parts per million to cc STP of given gas specie per cc STP of the total gas. This would be better qualified as a unit change and would be associated with a certain weight in contribution to the total change to the dataset.

CHAPTER 5
DATABASE VERSIONING

The RRUFF Database is "an integrated database of Raman spectra, X-ray diffraction and chemistry data for minerals" (RRUFF homepage). One of the features of RRUFF which is pertinent to this document is that RRUFF provides a web accessible changelog. As rows in specific tables are changed, these changes are reported through the log. The framework for communicating change information resembles the spreadsheet context. However, the main difference is the method to refer to the Attributes as they are mapped across versions. Database tables do not have strict identifiers for rows and columns as spreadsheets do. Row identification relies on indexed keys to uniquely identify entries, but these entries can also be arranged and presented in different orderings depending on the queries used to view the database. From Reference 24, elements within the database do not have to be digitally organized in the same way that it is physically presented.

Databases are also meant to be kept online for extended periods of time. Spreadsheets must be entirely republished in order for a change to the data to be made public. Changes in a database are therefore more sparse in between each version. In order to make changelogs more comprehensive, they can describe changes by day or other increment of time or describe the latest particular subset of time.

Once the changes within the database are encoded into the versioning conceptual model, publishing the changes can follow the methods detailed in Chapter 4

CHAPTER 6
ONTOLOGY VERSIONING

The Global Change Master Directory is a repository to find Earth science data from NASA. They employ a strict set of keywords to tag and label datasets in order to make them more searchable. Version 1.0.0 of GCMD Keywords was published on April 24, 1995, and as of the time of writing, the most recent version of the keywords is 8.4. As can be seen, the naming scheme of the versions changed since the first publication of the keywords. In the initial scheme, each part of the decimal system represented a different level of the GCMD Keyword hierarchy. When a change occured to a concept in a level of the hierarchy, the associated version number increments.

Since there exists an implementation of the GCMD Keywords in RDF, the URIs can be used as references for the Attributes in the concept model. As mentioned in Reference 8, in order for a data consumer to understand how to use a new version of an ontology, they need not only understand what concepts are new and what concepts are old, but also how to map the old ontology onto the new ontology. The mapping then informs the migration of Keyword labeling of datasets between versions.

CHAPTER 7
CONCLUSION

REFERENCES

[1] B. R. Barkstrom and J. J. Bates, "Digital library issues arising from earth science data," 2006.

[2] M. D. Flouris, "Clotho: Transparent data versioning at the block i/o level," in *In Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST 2004)*, 2004, pp. 315–328.

[3] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *Proceedings of the International Conference on Software Maintenance*, ser. ICSM '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 23–32. [Online]. Available: `http://dl.acm.org/citation.cfm?id=942800.943568`

[4] S. Chacon, *Pro Git*, 1st ed. Berkely, CA, USA: Apress, 2009.

[5] P. Cederqvist, R. Pesch *et al.*, *Version management with CVS*. Network Theory Ltd., 2002.

[6] W. F. Tichy, "Rcsa system for version control," *Software: Practice and Experience*, vol. 15, no. 7, pp. 637–654, 1985.

[7] J. Dijkstra, *On complex objects and versioning in complex environments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 13–23. [Online]. Available: `http://dx.doi.org/10.1007/BFb0024353`