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)

© Copyright 2016
by
Benno Lee
All Rights Reserved

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

ABSTRACT

The growth of data driven technologies has "exploded" the need for reliable, high quality data. This data powers the science of major agencies like NASA and laboratories like CERN. The changes made to maintain its quality has serious implications to the quality of the data that system uses as they propagate down stream through a workflow. Provenance plays a large part in identifying factors that contribute to data change. Technologies like PROV and OPM have allowed researchers to instantiate and track the entities and activities which went into generating their data sets. However, these ontologies do not cover change information in detail, and arguably, this falls outside their purview. Provenance discloses the contributors to the generation of a data object, but versioning describes the relationship that object has with its previous or future iterations. Changes to an object's provenance undoubtedly creates a new version of that data, but measuring the magnitude of that change still remains difficult. Current methods to quantitatively determine the distance between versions of a data object often involve comparing provenance graphs, but these approaches lacks the detail to make meaningful comparisons. There is a gap in understanding the transition from an old data set to a new one, and developing a more detailed understanding of change information allows users to comprehend how a data set evolves over time.

Much work has already been accomplished towards filling in this gap and laying the foundations to further address issues in this area. The first part of this thesis presents a concept model developed as a foundation to capture changes across versions. It does so by capturing the relationships involved in the three core versioning operations: addition, invalidation, and modification. Two data sets stored in Excel spreadsheets were used to develop and test the versioning model with the goal of describing changes in a scalable way that can expand to other applications. In open source software, changelogs provide more detailed change information to users and developers as a documentation artifact. However, changelogs traditionally present its contents only as human readable formats text. This work adopts changelogs as

a documentation tool to communicate change, but also adopt semantic web technologies by encoding machine readable content into the log using RDFa. In such a way, the changelog can be provided as an openly available HTML page, required to use RDFa, which is also machine accessible.

Work needs to continue to completely demonstrate the utility of the versioning model and its applicability. Much of the model construction and distribution has already been demonstrated with spreadsheets, but to ensure flexibility and applicability, other contexts must be hooked into the model. Databases provide a unique challenge because transactions do not create new instances of the data object. The versioning of ontologies provides significant insight into the growth of vocabularies to a domain as well as expand the application of older data sets. GCMD keywords do not constitute an ontology, but changes to terms within the hierarchical vocabulary have significant impact on the searchability and discoverability of Earth Science data sets. Applying the process of versioning the vocabulary could provide greater freedom to evolve without sacrificing the ability to find data sets. As a final endeavor, a method needs to be developed to utilize the structure of the resulting versioning graph in performing a flux-like calculation and give a quantifiable distance measure for the amount of change between versions of data.

CHAPTER 1
INTRODUCTION

John C. Maxwell once said, "Change is inevitable. Growth is optional." While this inspirational quote refers to the human character, it also holds true for scientific datasets. With changing technology, data collected by researchers grew at an astounding rate. NASA's Atmospheric Science Data Center (ASDC) reported a growth from hosting around five million files to twenty million files between 2001 and 2004 as seen in Figure 1.1 [2]. The ATLAS project at CERN reports that it generates on the order of four thousand new datasets per day from experimental tests alone shown in Table 1.1 [1]. The sheer volume of data generated per year by each of these organizations easily demonstrates the futility of managing these data archives manually. The desire and ease in which data transparency can be provided to not only researchers but also the public lies behind the drive of expanding the availability of high quality data holdings. The key to meeting these demands is automation, not only in distribution, but also in data quality management. However, these two dynamics are at odds with each other. Many NASA datasets have required re-processing of their data, either to improve data quality or to correct for errors [12]. However, we can also see that the number of distinct users doubled over the course of three years at the ASDC while the amount of data distributed more than triples. As such, the strain of informing and providing updated data to this body of users grows tremendously. The solution, thus, lies in passing on the ability to verify data quality to the users. Data traceability now becomes particularly important to identify sources that contribute to improved data quality. It creates a need to understand not only that a data set has changed, but to also understand how much a data set has changed. Data versioning is the method of tracking the changes performed on a data set and determining the extent to which it has changed. In this document, data versioning is approached using technology provided by semantic technologies and applying them to artifacts currently generated by scientific data sets.

Metric	2001	2004
Data Volume of Holdings [TB]	340	1,250
Number of Files	5 M	20 M
Data Volume Distributed [TB]	34	114
No. of Distinct Users	6,000	12,000
Production Jobs Run per Day	2,000	5,000

Figure 1.1: Summary of ASDC Holdings for the years 2001 to 2004 from Barkstrom and Bates [2]

Metric	Magnitude
Data per day [TB]	1
Data per year [PB]	20
Datasets per acquisition	O(4000)
Simulation Datasets per day	O(1500)
Versions per Dataset	O(1)
Files per Dataset	O(100)

Table 1.1: Summary of ATLAS data set generation in 2008 from Branco et al.[1]

1.1 Data Set Proliferation

An interesting statistic to note in Table 1.1 is the order of versions per dataset. While it does not indicate that the data have marked volatility, it does communicate that the data has a tendency to change over time. This means that not only does an archive maintain a data set, but it may need to maintain multiple instances of that work over time. Therefore scientists need the ability to discern between instances of the data they use when comparing results from other researchers. With the web's development, libraries and library sciences provide a steady evolution in methods to identify data collections. The challenges and goals that face physical libraries remain valid even as data collection migrates to electronic alternatives [13]. Digital storage and the Internet has opened new opportunities and methods to administer book data by separating logical representations and physical representations [2]. Publications, for instance, may be sorted and rearranged in a digital view along a

wide array of characteristics to provide the most logical presentation for searching. Comparatively, users search a physical collection by conforming to a rigid organizational system adapted to the provider's needs, not the user. It has also added a plethora of new content types such as wikis, blogs, and other document formats which have never seen physical print. All these new documents need a form of data management [14]. However, the migration has not been without its problems. Early citations used stagnant Uniform Resource Locators (URL) to refer to online documents, but this would lead to a condition known as link rot where moving the document would invalidate the URL [15]. This eventually led to the development of Persistent URLs (PURL) which also succumbed to link rot, and this eventually led to the distributed Digital Object Identifier (DOI) system used to track documents today [3]. In the table taken from Duerr et al. [3], DOIs represent the most suitable identifier used for science. It's origins begin in the Handle system, which can be seen as a generalization of the DOI system. The DOI network provides a robust system to track documents, but when tracking data, it faces difficulty following the rate of change with some more volatile data sets. Distribution organizations assign a DOI whenever a new edition of a document becomes available, and due to the publication process, documents change very rarely so a new DOIs are rarely necessary. However, data sets are products and thus succumb to the iterative process of error correction and growth. Data collection often continues on after initial publication. DOI distributors treat new files like new sections to a paper and changes to files as edits so a new identifier must be issued to the data set. This behavior becomes entirely too slow as data providers begin to allow users to dynamically generate data products from existing data according to their needs [16].

With respect to Figure 1.2, no identification scheme fits the description of a scientific identifier. Duerr et al. define a use case to make the argument that scientifically unique identifiers are necessary, "to be able to tell that two data instances contain the same information even if the formats are different" [3]. This highlights the necessity of being able to discern between the logical content of a dataset and the physical form of that object. As identifiers often use physical characteristics of the data to make comparisons and determine similarity, a proper representation

Table 2 Suitable identifiers for each use case where solid green indicates high suitability, vertical yellow stripes indicates good to fair suitability; and orange diagonal stripes indicates low suitability

Identifier Type	Unique Identifier		Unique Locator		Citable Locator		Scientifically Unique Identifier	
	Dataset	Item	Dataset	Item	Dataset	Item	Dataset	Item
ARK								
DOI								
XRI								
Handle								
LSID								
OID								
PURL								
URL/URN/ URI								
UUID								

Figure 1.2: Table of predominant identifiers used in science. From Duerr et al. [3]

becomes difficult. However, a model recently released model by the Health Care and Life Sciences (HCLS) Interest Group may provide a solution when used in conjunction with other identifiers [4]. Their model, shown in Figure 1.3, separates the concept of a dataset into three parts. The highest level summarizes the data as an abstract work, perhaps better described as a topic or title. This data topic can have multiple versions as it changes over time. The version can then be instantiated into various distributions with different formats. Such a set of linked data would provide sufficient data to discern between two data instances of the same information. However, it is unclear as to whether a single identifier would be able to encompass such complete information.

The organization of the model has interesting parallels with Plato's theory of Forms where he proposes that there exists a perfect idea of an object which has imperfect realizations in the physical world. Likewise, the summary description of the data set represents the ideal which the data seeks to capture. The model then uses a series of versions represented using the Provenance, Authoring, and Versioning (PAV) ontology to document the physical change of the data as it approaches the ideal[17]. PAV produces an interesting entry point into explaining the data set's development as well as recognizing the imperfection inherent in data capture. As

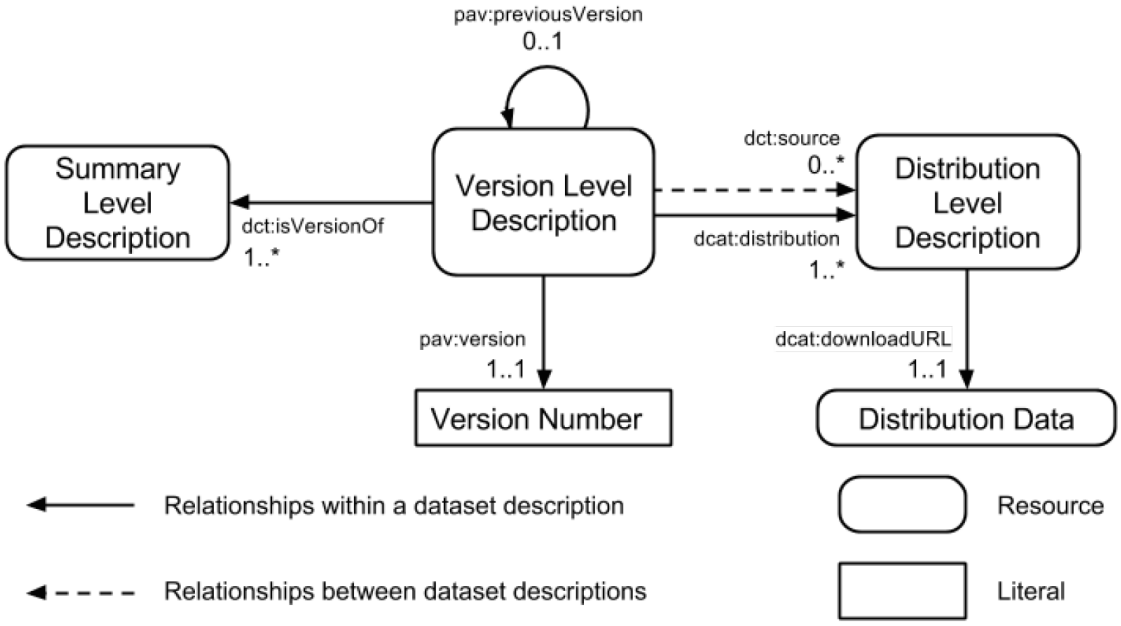


Figure 1.3: Data model from the Health Care and Life Sciences Interest Group separating data into three levels: works, versions, and instances. From Dummontier et al. [4]

a concept in the model, linked data can further extend the description to provide details necessary in identifying a particular version of data. However, PAV only has retrospective views of data versions since a version can only point back to previous instances.

For similar reasons, treating data as documents produces problems when applying technologies from software management [11][18]. Structure provides the most significant distinguisher between data and software since a data set with a removed file remains usable but a software project would break. The function of code comes from its content, but the function of data comes from its ability to store and organize data. This should not be confused with data formats which impose structure onto data in much the same way programming languages provides a medium to express actions. However, exporting data in different formats is currently easier than exporting code into different languages. Data sets do not represent a single object, unlike a software project[5]. They are compact representations of all possible subsets of the data set, which are also datasets. The Atmospheric Radiation Measurement (ARM) Program publishes data daily from sensors deployed across the globe, but a

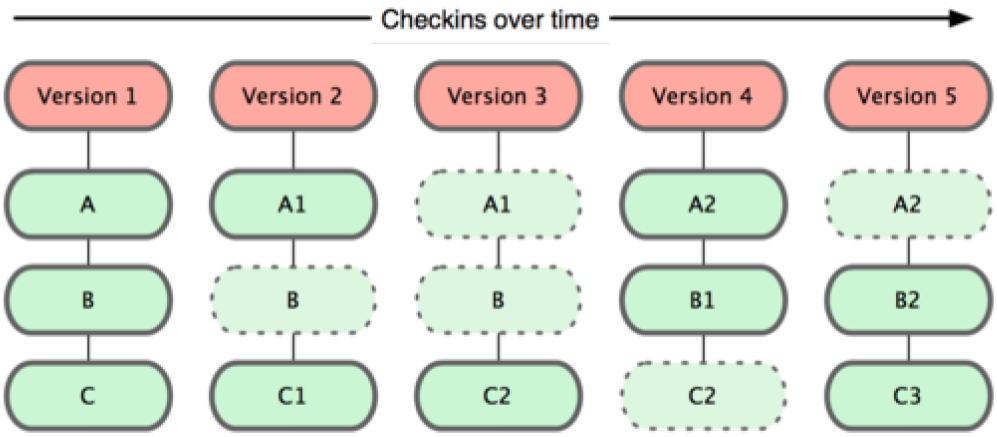


Figure 1.4: GIT stores changes in the repository as snapshots of individual files. Figure 1.5 from [5]

user may only desire files from a specific region, files from only the year 2012 in the Southern Great Plains region, or from only the month of February for the collection site's lifespan [19]. These can all be considered arbitrary subsets from the data sets generated by the data collection program, but a software project would not be able to sustain such arbitrary filtering. GIT stores the files comprising a repository as snapshots of each file in the store, and when developers make changes to a file, the store creates a new snapshot of that files as seen in Figure 1.4. A version then becomes defined by the complete collection of snapshots within the repository at the time, and the files cannot be subsetted due to the dependencies between the files in the version. As a result, when modeling the commit history, or the history of versions, the set of files can be compacted since all components are necessary to produce the object. Demonstrated in Figure 1.5, a developer creates a branch of the master line with commit C3 from C2. The entire repository at C2 gets copied into the branch, and when the work is done, it gets merged back into the master branch. When a user then orders the software generated by this repository, it results from the compilation of the entire commit whereas a data set does not require the same completeness to be used. For this reason, the structures of data sets and software becomes incompatible and software versioning technologies are insufficient to capture this nuance.

Furthermore, returning to Figure 1.4, GIT actually can be considered a re-

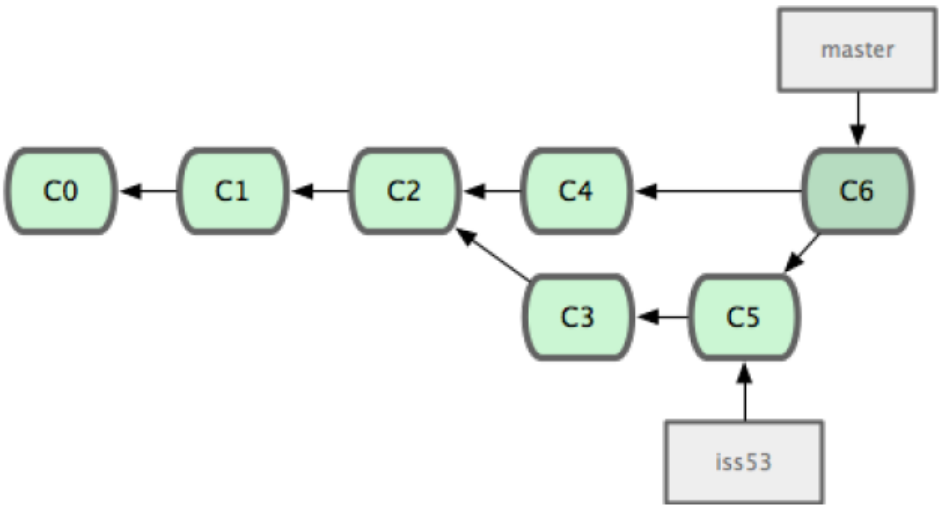


Figure 1.5: Example of a commit history with branching stored in GIT. Figure 3.17 from [5]

duction in terms of software versioning methodology. The constant dynamic in play with tracking versions, perhaps with Computer Science in general, is the trade off between space and time. Maintaining full snapshots of repositories proves extremely space consuming, but retrieving these snapshots occurs near instantaneously. In general, prior to this, the conventional wisdom was to instead store one snapshot and a set of deltas or differences between software documents and then use processing time to reconstruct a particular commit by applying the differences [11]. Deltas can further break down into forward or backward changes, reflecting the importance of keeping the snapshot of the oldest or newest instance. This produced very lean versioning systems, but after a long string of changes, applying deltas becomes very time consuming so periodical snapshots become necessary. With costs for storage space compared to the relative size of software files dropping, GIT prioritizes the ability to quickly swap between and develop different branches of the system. This interplay between space and computation time can also be seen within the data management space as keeping snapshots of large databases will be intensely costly but less so for spreadsheets.

The techniques employed by these technologies, however, can remain applicable to data sets and are often necessary when communicating change data to users. Version producers often refer to versions using numbers in the dot-decimal style [20].

While the values often signify the Major-minor numbers associated with the version, the names remain meaningless and can arbitrary assignment such as Ubuntu released numbered by Year-month values [21]. The arbitrary nature of the numbers often entails referring to versions by English nicknames instead. Such a regular method of naming release versions also means that determining the magnitude of change between two releases becomes impossible. Numbering the version this way, however, does allow computers and readers to quickly parse the version name and discern that a change has occurred, but little value exists beyond that [22]. The technique of distributed and federated employed by GIT does provide significant value to modern methods of versioning data [23]. As data workflows and data set dependencies grow, their volatility also expands, meaning that they become more likely to generate new versions. The federated approach available in the GIT environment allows developers to establish change dams that collect modifications and releasing the data at regular intervals,reducing the changes to a manageable flow.

1.1.1 Unified Systems

Working with data as documents leads to the shortfall of technologies, but working with the data of documents has led to significantly greater contributions. Many libraries often work in collaboration in order to provide a wide selection of texts over a limited number of physically available documents. The University of Virginia demonstrates the ability to achieve a unified library system using a combination of XML and web service technologies of their disparate assemblage of libraries [24]. The challenge involves providing a common landscape in order to compare the quality requirements imposed on the repositories. Versioning systems provide a notable mechanism to make this decision as quality determines when to generate new versions and what items belong to the same groupings of data. The comprehensiveness of XML and web technologies also allows this approach to apply to other systems and research areas as well. This becomes particularly relevant as innovations in computation technology generates small, volatile data sets to integrate into larger data managers [25]. In this application, the data food chain then becomes represented by smaller applications generated in situ and then

unified with other data sets as they move up the food chain to a large, unified data distribution center.

Unified libraries represent a part of a larger collection of systems that rely on the propagation of data through heterogeneous systems to produce rapid complex solutions. The grid provided a unique environment that had to handle a variety of inputs, and therefore, different input data could run on distinct sets of grid services. This meant that different versions of the same data could be generated by differing services on the same grid [26]. The CERN grid for the Compact Muon Solenoid experiment separates the physical and logical storage of files, allowing multiple users to refer to the same file without needing to copy the file across the grid [27]. While the structure and construction of the grid reduced the uncertainty introduced by varying hardware, it raised questions on data quality by abstracting the transparency to underlying services. Cloud services have recently replaced the grid due to its flexibility in the services available to its autonomous systems. As the scale and complexity of autonomous systems grow, it becomes more difficult for one system or organization to manage all the circles necessary to produce data deliverables. The ability to propagate relevant data change data across autonomous systems then assures valid quality in interactions between domains [28]. Not only does this ensure uniformity through system interaction, but it also ensures transparency with respect to the data and methods used to produce conclusions [29]. This often means that systems will need to negotiate a contract and establish a mutual interface to exchange data. Occasionally, this contract can be formal, but more ideally, the establishment of a standard lineage model or format would allow a greater variety of systems to interact with each other without needing lengthy contractual exchanges.

1.2 Defining Versions and Versioning

Researchers often use the words version and versioning without needing to explicitly defining them because they appear ubiquitously in data management systems. As a result, the definition of a version varies depending on the application, but they all try to capture a common idea. Barkstrom describes versions as homogeneous groupings used to control, "production volatility induced by changes in algorithms

and coefficients as result of validation and reprocessing," [6]. He separates groupings by the granularity at which observations are collected. This construction allows users to easily determine the similarity or dissimilarity of objects within the grouping. However, NASA employs a rigid data processing structure with well defined levels. This means that the resulting hierarchy is well balanced.

Another application defines a version as a "semantically meaningful snapshot of a design object," [29].

It is more constructive to define versions as a matter of purpose. When two objects are versions of each other, they are considered to still serve the same purpose. For data production services, this would translate to there they exist in the workflow. Two objects must, therefore, share provenance, and the more they share, the more likely versioning practices will provide meaningful results.

Versioning is the activity of exposing changes relating two or more versions of an object. Versioning differs from provenance because an activity does not establish an object as a version of another even though the activity may create it. The relationships exist as a matter of state-based properties which are exposed through versioning. These relationships have greater significance if the objects are versions of each other.

1.3 Data Quality/Provenance

The fundamental challenge to determining data quality, its subjectivity, needs clarification. Conceptually, a data set on desert climate likely has very poor data quality and relevancy to a study in whale biology. However, in a more quantitative sense, that same desert climate data can have excellent quality with respect to its correctness, expression, and traceability. With the hybridization of data sets from disparate agencies to provide big data solutions, collaborations plays an ever present role in achieving broad, valid findings. This requires good quality data and the ability to determine when data with better quality becomes available. [30]. The primary focus, generally, involves tracing the lineage of artifacts and activities that lead to the current data. This provides insight into possible sources of error as well as validating the assumptions made in generating a data set for future use.

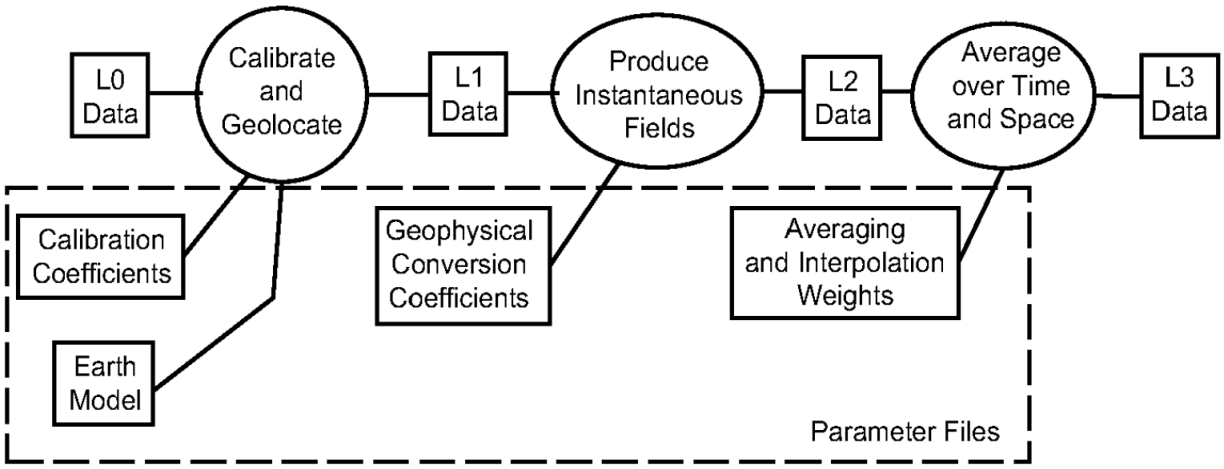


Figure 1.6: NASA organizes its data into three levels depending on the amount of aggregation and the distance the data is removed from the original sensor measurements. Figure 1 from [6]

There are several characteristics that can describe a data's quality, but the one most relevant to data versioning is provenance. It describes the sequence of events that lead to the construction of an object [31]. In art this describes the sequence of people who have had ownership of a piece of artwork. For data, provenance relates the history of inputs and operations that result in a data object such as a plot or data set. NASA defines three levels of data processing, seen in Figure 1.6, that encompass the stages required to turn a raw signal from satellite instruments into physical measurements into global aggregate summaries [6]. Each stage computes a dataset using a collection of input data, processing scripts, and calibration values. This collection forms the provenance for the resulting level of data, and it contributes to the production history of the higher level products it feeds into. More specifically, scripts and code which "Produce Instantaneous Fields" operates on "L1 Data" using "Geophysical Conversion Coefficients." The diagram clearly illustrates that should "Calibration Coefficients" used to generate "L1 Data" change, the resulting data would cause "Produce Instantaneous Fields" to yield different "L2 Data." As such, strong ties lie between identifying versions and locating differences between provenance changes.

As data sets grow, this process becomes even more confusing to coordinate so version control systems often manage provenance. Current research endeavors

to provide high quality data clearly becomes more formalized as data becomes concentrated in massive data warehouses [32]. The focus of a majority of versioning research focuses on lineage retrieval which becomes ever important as evidence grows that researchers generate data faster than they can reasonably track [33]. This poses a particularly difficult problem as provenance provides a potent means of data auditing. With provenance, data producers ensure the trustability of their data inputs, either ingested from external sources or integrated from internal data sets. Fairly reassuring results have been found when combining lineage management and error reporting systems [34]. The errors provide a context for the changes made to advance the lineage of the data set and the version manager demonstrates that a problem has been addressed and how it was corrected. This system becomes extremely important when considering that agency funding often depends on the ability to account for the value of a project's dataset [35]. The data analytics required to determine the value of data collected by a project also requires the provenance to ensure that the analysis is also reproducible. The basic provenance often collected by hand now needs to be collected automatically in order to facilitate collaboration, especially with projects that are farther away from the data.

Early attempts at encoding provenance data into semantic models include the development of the Proof Markup Language [36]. While this was originally developed to express inference reasoning through traceable graph relations, the model can also be used to express the provenance of products using the same transitions. The power is that it is able to use terms defined on the Semantic Web to construct inferences. This early demonstration of the ability for web based semantic technologies also expresses complex relations in a way that can be reasoned over and computed. It then allows for autonomous solutions to understanding change as data freshness begins playing a significant role in successful system function [37].

Not long after began the development of the Open Provenance Model (OPM) [38]. Driven by the uncertain needs and sometimes conflicting conventions of different scientific domains, the model sought to find a method to standardized the way in which provenance data is captured while also keeping the specification open to accommodate current data sets through the change. In an experimental case, the

model has been applied to sensor networks to automate and unify their provenance capture even as they grow [39]. To aid in the adoption of the OPM, the framework Karma2 was developed to assist integrating provenance capture into scientific workflows [40]. It reduces the amount of modifications required to adopt the OPM through web services and, more importantly, integrates into scientific workflows. With the magnitude of data collection endeavors, it is no longer feasible for scientists to stay close to the data and must take a more abstract view of their data collection activities. Scientific workflows provides this high level view of complex data collection, curation, and analysis [41]. The value then of integrating provenance capture and workflow design is that lineage planning can then take place at a high level of scientific work. This gives insight into how different parts of the workflow fit together and how new exploratory expansions may occur.

Following the OPM, PROV is a W3C recommendation that deliniates a method to express data provenance with semantic technologies that has been accepted as a World Wide Web Consortium (W3C) Recommendation [42] [43] [44]. The recommendation uses a conceptual model relating activities, agents, and entities together to describe data production lineage [45] [46] [47]. Intended as a high level abstraction, it describes data as entities that are generated by activities enacted by agents. This basic relationship is very powerful in its ability to describe data production activities. The expression of the conceptual model occurs through the PROV Ontology (PROV-O), which can be conveyed through various resource description languages [9] [48] [49]. The ontology is further formalized into a functional notation for easier human consumption [50] [51]. One particular strength that has contributed to the adoption of PROV is its ability to link into other ontlogies, making it easier for existing semantically enriched data sets to adopt PROV [52] [53]. Like the OPM, a framework has also been developed to alleviate workflow integration through Komadu [54]. The framework improves over its predecessor, Karma, by no longer utilizing global context identifiers that were no necessarily shared throughout the workflow.

The PROV Ontology provides four different concepts that begin to encapsulate the provenance relationship between data versions. The ontology defines a

prov:Generation as "the completion of production of a new entity by an activity," [9]. This means that the generation must result from a prov:Activity. In versioning, activities play a much less active role since changes become exposed from comparing like objects. It creates a relationship between an entity and an activity, but such a relationship may lead to an implication that a change in the activity resulted in changes in the resulting version. Changes could also result from modifications in the input data, leading to an entirely new generating activity rather than a modified one. Prov:Invalidation likewise makes a similar connection between activities and entities. This means that PROV-O does not have the direct means to communicate the addition and invalidation relationships which exist in a versioning context. Since versioning relationships result from state-based effects between version entities, a more contextually appropriate relationship would connect two entities together. The property prov:Derivation does relate together two entities and the ontology defines it as, "a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity. " [9]. In the case of the MBVL dataset, none of these three assertions hold true. The four versions are being simultaneously considered so one is not being transformed into another as would a sequential set of versions. Additionally, since we do not know which version is the best, we cannot consider any version an update of the others. Finally, no entity pre-existed as the data sets resulted from an ongoing analysis and further steps have not been developed.

PROV has played a significant contribution in maintaining the quality and reproducibility of datasets and reporting in the National Climate Assessment (NCA) [55]. This implication signifies that there is an increased likelihood of adoption through other scientific fields as a result of this reporting. The Global Change Information System, which houses the data used to generate the NCA, uses PROV to meticulously track the generation of its artifacts and results as they are used in the report [56]. This means that not only does the data have a traceable lineage to verify quality, but the content of documents can have the same verifiability [57].

1.3.1 Changelogs

Changelogs, sometimes called patch notes, are artifacts resulting from the versioning process often found in major software projects. They document the changes made within the system and seek to explain, in human language, the motivations behind changes [58]. The logs provide significant utility to both users and producers as it can serve as both documentation and tutorials. Many users will often refer to the patch notes in order to decide how to adapt to changes made to the system they use, either data or software. Meanwhile, changelogs aid producers through team transitions by keeping a history of decisions made to improve the project. This is particularly evident in the realm of open-source projects as developers can contribute without having been part of the original development team. The need for documentation to bring new programmers up to speed for a project drives the ability to keep the project alive.

Open source projects have much more consistent adoption of changelogs than data sets, possibly resulting from complex code techniques emerging earlier than large data methods. 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 [59]. These projects have a tendency to die rather quickly after initial enthusiasm and with the rather low overhead cost to start new open-source projects, some automated methods of determining the progress of a project is needed. It would give insight into the maturity of a project's development team as well as the likelihood that team members will correct errors within the code. However, readability proves to remain a significant hurdle as current development change logs contain solely human readable text. While machines may still be significantly removed from the ability to comprehend the impact of changes made to a data set or software code, they are currently opaquely blocked from consuming any of the content within logs more than understanding they contain text. 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.

As mentioned previously, changelogs also allow developers to link bugs and errors with their corrections in new versions of the code [60]. This gives feedback to

the user community that corrections have been addressed as well as ensuring that modifications to the code base are driven by improving the project. It also has the added benefit of creating a system that can be used to link the introduction of new features with the emergence of new bugs [61]. The resulting discoveries help reveal patterns of development and prevent further occurrences of problematic code. Therefore, providing an machine consumable changelog would accelerate and assist in navigating through dataset changes and error corrections.

1.3.2 RDFa

In order address the human readability of data change logs, this project considers the use of the Resource Description Framework in Attributes (RDFa) framework [62]. Figure 1.7 illustrates the semantic difference between what web crawlers and what humans see when they consume web pages. People intuitively understand that certain strings represent meaningful information, and RDFa seeks to encode that understanding natively into the document in order to allow them the ability to consume the information more effectively. The framework leverages the existence of various established vocabularies in order to provide a standardized understanding of web documents, and it opens the ability for them to interact with the web pages more intelligently. The benefits of embedding RDFa into change logs is twofold. First, the change log would need to be marked up in HTML in order to accept RDFa. As a result, the log would also become available on-line and thus, more openly accessible to data users through the utilization of web based search engines. This would allow data users to better determine personally how a change applies to their specific application. Large companies such as Google have already begun making endeavors in equipping their web crawlers to consume structured data such as RDFa from web pages. Second, the simple application of RDFa attributes encodes the entries within the change log in a format consumable by machines [7]. RDFa has already had significant success in adoption across a variety of web publication platform and eases the search for their content [63]. In these applications, however, the developers use RDFa to describe the content on the page, to indicate a string is actually a name for example. This project endeavors to use RDFa to embed an RDF

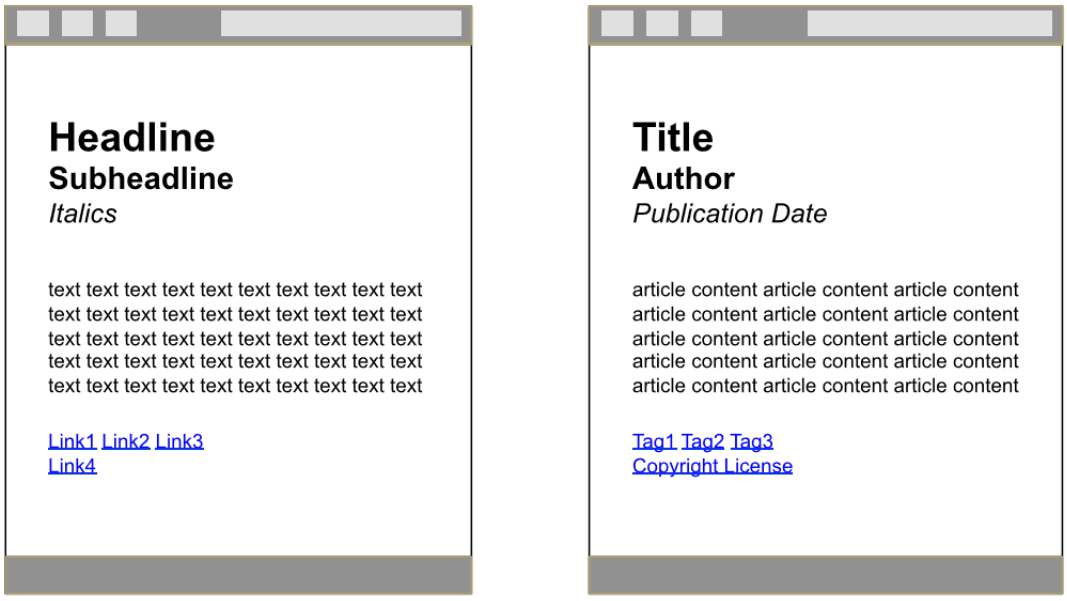


Figure 1.7: Illustration of the difference in what autonomous systems see when crawling a web page and what humans see when reading the same material. Figure 1 from [7]

graph into the web page instead, and therefore, the data becomes captured instead of described. The language does have the ability to transform into RDF, but the slight nuance between intended use means that a more complicated deployment of the attributes will be required. Using a previously established standard eases the adoption of encoding required to communication change information to autonomous systems.

1.4 Provenance Distance

Understanding provenance and workflows only provide only a portion of the view into a data set's lineage landscape. The workflow provides an understanding of how a data set fits into the bigger picture of data analysis and the provenance gives a method to reproduce the data set for data quality purposes. As such, workflow and provenance describe data sets in a very flat and static manner, allowing

for prospective reasoning as to how it may respond to changes made by the data producers. However, this places the burden of determining the magnitude of change in quality, as changes in provenance mean changes in quality, on the data producer. Consider again that data quality is subjective with respect to the data consumer's usage, and the difficulty in determining the significance of a new version becomes apparent. With increasing complexity, data workflows have developed in such a way that even subtle changes have serious implications for other parts of the workflow [10]. The responsibility of determining and communicating the magnitude of data alterations falls to versioning systems.

A very rudimentary way to communicate change distance uses the version number of the data set. Returning to discussing the dot-decimal notation often used to identify versions, version numbering follows a hierarchical method of systematically counting the releases made to a data set or system based on the perceived magnitude of the change. The problem lies with identifying the extent of perturbations performed upon the data set. The primary function of the number is to indicate compatibility not changes. Therefore, a release can result from five perturbations or fifty modifications. These cases become challenging since some users will experience larger perturbations resulting from a change than others. As a result, using fewer well-established categories avoids this problem, but it loses many details in resolution of the extent to which the data set may change. In addition, there is no standardization as to what each of the numerals used in a version number represents, and this significantly hinders interoperability between information systems. Data managers will often discuss whether data sets are qualitatively different enough to warrant incrementing one of the version numerals. While the dot-decimal method is easy to implement and use, its broad categorization severely impairs its ability to express version information beyond a basic functional extent.

Another approach is following the provenance of two data sets and identifying differences between the lineages of the two data sets. The total difference between the data is known as their provenance distance. This distance measure is very new as the availability of computable provenance has been developed fairly recent. One endeavor to compute over provenance has shown a marked ability to predict

disk usage based on the lineage of a data object [31]. Efforts have also been made to summarize provenance representations to improve consumption [64]. While the ability to compute over provenance data has been demonstrated, the comparison of two provenance graphs has yet to be widely studied.

Using PROV to represent provenance data in a semantic model produces an acyclic directed graph with labeled nodes. As a result, the provenance distance problem reduces to the similarity measurement problem. When measuring similarity, algorithms determine how far two graphs are from being isomorphic [65]. General graphs have similar complexity to determine similarity, but node labeling simplifies this process by providing a method to match nodes together. Other methods also exist to determine similarity under different conditions such as edits necessary to transform one graph into another [66]. Some methods focus primarily on edge changes [8]. In Figure 1.8, the left graph transforms through a move of edge 1 and a rotation of edge 4, resulting in an edit distance of two. Such changes in a provenance graph would demonstrate a change in dependencies between objects used to generate a final product of note. A difference in provenance distance would then provide context for differences between conclusions. Similar results from data with small provenance distance would ensure reproducibility, while large distances signals reinforcing confirmation of a result. However, differences can always be expected when considering different versions with more detailed comparisons required to determine the implications of changes made to a data set as a result of a provenance change. Meanwhile, when results disagree, a small provenance distance indicates that findings may not be reproducible. This kind of analysis resembles comparison measures employed in determining semantic similarity [67]. The main difference lies in semantic similarity comparing the distance between two concepts within a graph as opposed to the distances between the graphs themselves. However, it does reveal that using semantic graphs can have incredible impact in extracting implied relations between data they store.

There already exist methods which compare workflows based on quality criteria that leverages provenance to bound quality of service [68]. However, these procedures focus primarily on quick retrieval and efficient storage instead of lever-

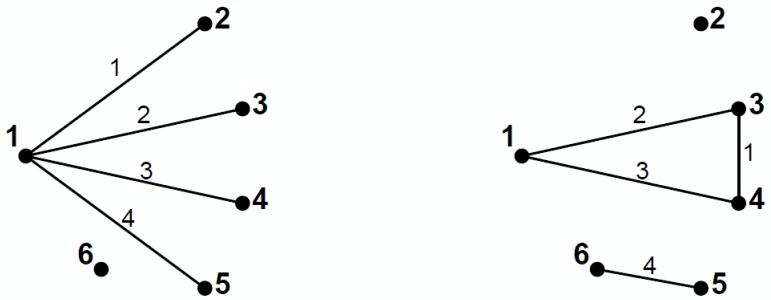


Figure 1.8: The labeled graph on the left transforms into the right graph under two edge edits. Figure 2 from [8]

aging the latent information accessed by reasoning across data set versions [69]. The distance measures previously mentioned rely solely on provenance graphs to compute results, but this is obviously insufficient. When considering the provenance of a data object, methods only consider the activities and entities that took an active role in the production of it. A new version of an object has a familial relationship with its previous versions, but in most cases, they do not take an active role in its generation. For this reason, detailed change information falls outside of provenance's scope and it can be seen in PROV using a single relation to link different versions of a data object. Without detailed change information, determining the difference between two data objects in a metric beyond broad strokes becomes difficult if not impossible.

This is not to say that provenance becomes useless in computing change distance, but it largely serves as an indicator than a measure. If there is any difference in provenance, then something must have changed. For example, if workflow uses a new script to generate a data set, changes can be expected in this new data set. However, what this script does differently than the old one requires a more detailed understanding and description than lineage can provide [33]. Additionally, if no changes were made to the script, but new data was produced, it likely indicates that some inputs have changed. The ramifications for the resulting data set will be difficult to determine without understanding how the original inputs have changed. Only knowing that they have changes is insufficient. Being able to understand the extent that modifications to data or workflows impact the results greatly improve a

producer's ability to generate high quality data.

1.5 Data Versioning Operations

Architecture has a principle that says form follows function, but, for data, form equals function. As a result, data has as many different forms as it has functions. Biological experiments often use data within cyclical data workflows where outputs are immediately fed back into new experiments [29]. Even though the goal of the experiment is the final data set, all the intermediary data sets provide significant value in reaching the goal. Libraries store data about their collections in large databases where both old and new versions of literature need to be maintained [30]. Some data exist in such a highly constrained environment that it must be managed at near the hardware level [70]. The struggle no longer becomes generating data, but instead, fitting the data into a format that users find useful and can consume.

The challenge of data versioning systems is to provide a unifying environment that can handle the plethora of forms and functions of its data. At its core, versioning systems only need to concern themselves with three operations: addition, deletion, and modification. Most literature surveys do not realize the significance of this commonality as this means that versioning methods can be described by delineating how each operation is approached by a system [29] [13]. Data addition generally constitutes the least complicated versioning operation because it interacts the least with pre-existing data. However, new data does share context with pre-existing data and provides a method of measuring data set growth. Since data sets no longer have to be used in its entirety and can be freely subsetted, a data set's complexity increases significantly with its growth. Every new file added to a data set doubles the number of available subsets.

Data deletion, however, has a more philosophical difference between systems. From the perspective of a versioning specialist, data should never be deleted since knowing why data was excluded is as important as knowing why data was included. The software versioning manager GIT uses a method of compressing older data to conserve space without deleting the data [5]. Pragmatically, this is not always possible due, generally, to the physical constraints of storage space. In high energy

physics, observational data often cannot be re-collected due to cost, and as a result, poor quality data cannot be re-processed or replaced [35]. The decision in this document is to use the term invalidation when referring to data removal operations as it implies that whether permanently deleted or not, there exists a more valid alternative.

Data modification encompasses the most involved data versioning operation. As a result, it often comprises a majority of the description of a data versioning service. In truth, data modification can be summarized as the invalidation of an instance of a data object - which can be a file, a record, or anything in a data set - followed by the addition of a new instance of that data object. However, this kind of operation is used so often to fix errors and update data sets that it is considered a unique operation. Modification owes its complexity to interacting with both pre-existing data from the invalidation stage and new data from the addition stage. However, this compound relationship fully contextualizes the relationship the operation has in relating the old data and the new data. In some cases, this only provides forward or backwards references between data versions, but having both gives users context for data's current state and update to new data [71].

Due to the ubiquity of the data addition, invalidation, and modification operations in versioning systems, the conceptual versioning model presented in Chapter 3 centers around capturing the relationships established by each of the operations. While other functions exist commonly in versioning systems such as object locking to prevent simultaneous conflicting changes, viewing to see the version an object belongs to, and branching to allow distributed modifications, these functions comprise the space of utility operations that support the three core processes.

1.5.1 Types of Change

The study of versioning operations further breaks down into categorization of change types data sets may undergo. While the meaning of operations are fairly easy to understand, not all changes have the same impact. As mentioned previously, version numbering separates perturbations into categories based on the impact the producer believes it has on the project. In this project, changes are categorized into

scientific, technical, and lexical changes. The granularity of the categorization does not consider the magnitude of change within the individual values stored by a data set as actual values vary depending on application and domain. Focusing on a more abstract representation of kinds of change allows for a better understanding of its impact while not being too precise to be domain specific.

Scientific changes comprise the family of changes which have the greatest impact on a project or data set. It indicates that modifications have been made to the most fundamental methods used to create a data set. These can include changes to algorithms used or sampling methods which may require a change in how users consume the new data. These changes have the largest implications for data consumers as it can have serious consequences for the soundness of their results. However, these kinds of changes is not always caught on the production end of data generation. While very large modifications can easily be determined to produce a scientific change, more subtle changes or interactions can also have larger ramifications, and data producers may initially view this as a technical change due to data quality's subjectivity. 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. In one of the data sets used by this projects, concentration units were originally reported in parts per million and then in cc isotope ratios. This would constitute a technical change since the data presents the same scientific measurements but in a different manner. 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. While they have the smallest impact on results and conclusions, these changes can allow computations to be performed when previously missing data discouraged such behavior.

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. Producers tend to use versioning systems to ensure data quality of service through audits and recovery tools [35]. 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.

1.6 Thesis Statement

The growth of innovative data capturing and storage technologies has led to new challenges in properly tracking the changes stored data sets undergo. Researchers store data in a variety of formats, from documents to databases, but since the growth in project scope, many have relied on versioning methods from software management technologies to track their data's evolution. However, these techniques fail to properly capture the changes data undergoes because they do not take into account the impact that a data's structure has on its function. In order to maintain data quality, producers use provenance meta-data capture the series of activities and agents involved in generating a data entity. Emergent technologies and frameworks have been developed to digitally capture this information including OPM and PROV. This data can be used to give insight into the magnitude of change a data entity has undergone by comparing the differences in provenance between the two entities, but this can only be done in broad strokes without more detailed change data.

By looking at the versioning transactions operating on a data set, data consumers can have a better idea as to the extent their data changes. This thesis document develops a concept model to formally characterize the activities and relationships among data objects when transitioning between versions. The model would improve current discussion on data versioning by providing a common understanding of terms and activities, changing versioning from a rule of thumb chore to a valuable research activity. It then demonstrates the model's utility by applying it to tabular spreadsheets, and then generalizes it to other contexts. In addition, this con-

tribution addresses in more detail the problem of measuring change distance, which using only provenance data could not accomplish, by utilizing the graph structure of linked data. Through this process, it creates and makes accessible machine readable change logs which allows for clearer deductions when comparing the extent of changes. This enriches the version documentation process without developing new artifacts, but now allows for the introduction of automation into part of the data auditing workflow. In addition, the method reinforces the need for dynamic, publicly available change information as data becomes more flexible to accommodate research in more diverse fields.

CHAPTER 2
PREVIOUS WORK

A number of instances in the literature mentioned previously uses the term versioning and provenance interchangeably. Mayernik et al. also notice a similar phenomenon although they use the term lineage instead of versioning [72]. The version model introduced by Barkstrom in Figure 1.6 to organize NASA's satellite data collection actually refers to a simplified workflow describing the provenance used to produce each level of data [6]. The diagram does not compare objects from the same level since changes to contributing components are only used as an indicator for version change. In actuality, objects have much more complicated development structures than the one dimensional lifespan indicated by the transition from Level 0 data to Level 3. The PROV Ontology model in Figure 2.1 outlines more explicit inter-relations between data objects, and it provides a new dimension with which to consider the interactions of data objects. More specifically, it outlines the explicit process of an agent performing an activity using an entity to produce a new entity. In the context of the level system, an agent (either a program or individual) performs a "Produce Instantaneous Fields" activity using "L1 Data" to produce "L2 Data." However, higher level data sets rarely use only one instance of lower level data. Calibration values may result from daily readings collected from another data set, but the generality of the ontology allows these relationships to be explicitly expressed. A more realistic provenance graph looks like the one in Figure 2 of an ozone indicator in which a Level 3 object results from the interrelation of multiple lower level products. An interesting observation of note is that Tilmes remarks in 2011 [10],

> Consider the relatively common case of the calibration table, which is an input to the L1B process, changing. Even though the version of the L2 or L3 software hasnt changed, the data files in the whole process have been affected by the change in the calibration.

which Barkstrom already observes in 2003 [6]

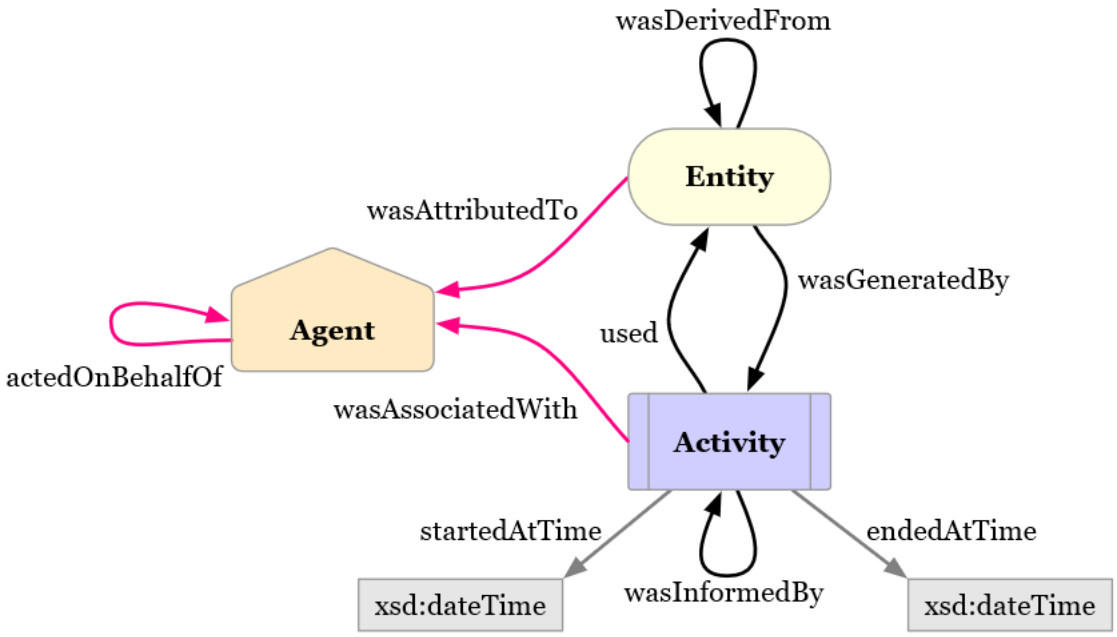


Figure 2.1: Diagram of the PROV Ontology. Figure 1 from [9]

> If scientific data production were easy, instruments would have stable calibrations and validation activities would discover no need for corrections that vary with time. Unfortunately, validation invariably shows that instrument calibrations drift and that algorithms need a better physical basis. Within a Data Set, we can think of a Data Set Version as a collection of files in a Data Set that have a homogeneous Data Production Strategy. Within a Data Set Version, we expect the code producing the files to be stable. We also expect that the algorithm input coefficients will be stable as well. The intent of data production is to produce data whose uncertainties are statistically similar under similar conditions of observation.

indicating a basic view that despite eight years in difference, the continuation of software focused versioning resulting in difficulties of data oriented collections.

If the level system provides a length and provenance indicates a breadth of a workflow, a version system can be considered to provide a height to a total workflow. Referring back to the HCLS data model terminology in Figure 1.3, as objects within a workflow, as in Figure 2, change versions, the structure of the workflow as well

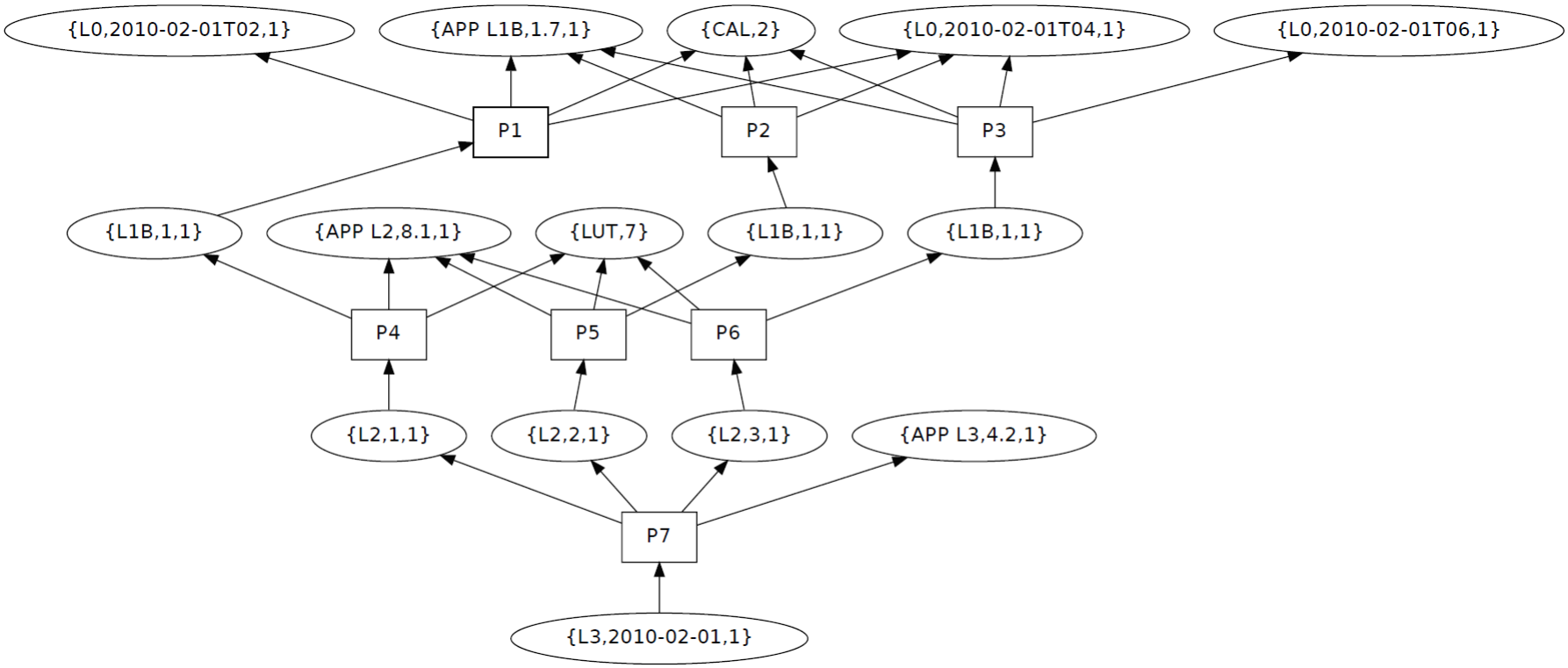


Figure 2.2: Provenance graph of a Level 3 data product, showing the inter-relations between different data products in generating the final product. Figure 2 from [10]

as the Summary Description of the final object, in this case the L3 Ozone product, remains the same. Instead, the new versions add layers like building blocks over the foundations of the original workflow structure. Version control systems then provide the mortar linking the blocks together to give the lineage capture procedure a solid structure. The PAV Ontology provides a means to track versioning information through linked data by introducing *pav:version* to cite versions and *pav:previousVersion* to link them together in order [17]. It does so in comparison to the Dublin Core concept *dc:isVersionOf* which records, "Changes in version imply substantive changes in content rather than differences in format" [73]. PAV argues that a new concept becomes necessary to cover cases where new versions do not have to be substantive but can still be alternate editions of the original object. Of note is the retrospective nature of the PAV ontology and PROV-O since it places primary emphasis on the most recent edition of an object. Figure 2.3, shows how RCS stores older versions as back deltas and branches as forward differences. The retrospective nature of back deltas results from development focus on the latest version, in this case 2.2. However, the forward differences provide a method to migrate from version 1.3 to the front of the branch. This characterizes the difference between a focus on data tracking, like that performed by provenance, to data migration, which users must undergo in order to consume the latest version. Mayernik et al. also find that, "Prospective records document a process that must be followed to generate a given class of products whereas retrospective records document a process that has already been executed" [72]. Retrospective provenance and versioning provides the ability to ensure data trustability and data quality among resources. However, researchers must follow a prospective versioning record in order to keep their research up to date.

To bring the discussion to an actual application, the GCMD released version 8.4 of their keywords, adding a slew of new values and modifying a select few [74]. At the time of release, many data repositories can be currently assumed to be using the previous or older versions of the keywords. As the taxonomy is not a class-based ontology, changes to the keywords have significant implications to the semantics of a data set described by those keywords. A data producer wishing to expose their

data sets using the new version of the GCMD Keywords must use a prospective method to translate their current descriptions to the new version. By comparison, the GCMD would use a retrospective measure to record the changes made to their keywords. PAV would not be able to address the prospective problem as a result of it is, "a lightweight vocabulary, for capturing just enough descriptions essential for web resources representing digitized knowledge" [17]. A detailed transition from GCMD Keywords 8.3 to 8.4 would significantly undermine the lightweight nature of the vocabulary. GCMD does provide a short summary of changes made in a version, but this would come in the form of text rather than structured data.

2.1 Spreadsheets

In this project, spreadsheets were chosen for study as they resemble text-like data objects while still maintaining a level of complexities. Though not as well encapsulated as other data format types such as the Hierarchical Data Format (HDF) or Network Common Data Form (NetCDF), spreadsheets provide many helpful tools that scientist favor for quick data storage and distribution over comma separated values (CSV). There also exists other document-like formats that are not discussed in this paper such as eXtensible Markup Language (XML). The initial work was done with the "Noble gas isotope abundances in terrestrial fluids" workbook (Noble Gas) [75]. The "Paragenetic Mode for Copper Minerals" workbook (Copper Data) was used to give better insight into data changes due to collaboration with the data

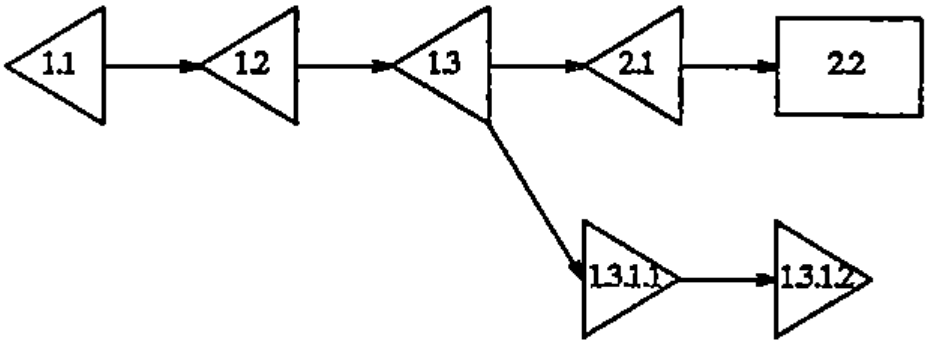


Figure 2.3: Commit history of an object in RCS with changes in the main line stored as back deltas and side branches stored as forward deltas. Figure 5 in [11]

set's author [76].

The Noble Gas data set was initially published on June 11, 2013 and then released a second version on March 8, 2015. Many significant changes were made to the data set between the two versions, which makes this data set particularly challenging to version. The physical structure of the data set changed from eight separate Excel spreadsheets to a single spreadsheet. The second version also trimmed 195 columns to 54 columns in the second release. In addition, many new locations were surveyed and added to the second release. Documentation accompanied the data set explaining different components of the spreadsheet and its usage, but it included no versioning information. This lack of versioning or transitioning information indicates a focus on data usage rather than data maturation, which is not a particularly bad approach. It makes logical sense to simply download the latest data set when it becomes available and not worry about the format of the invalidated data set. This approach convenient for new users of the data as the cost to consumer the new version of the data is the same cost they would have spent to acquire the data in the first place. However, users of the old data are disproportionately effected by the change in versions since old code and workflows may need to be updated to accommodate the changes in addition to the cost of consuming the architecture of the old data set. In this case, users would need to read the documentation to understand whether 182 from the June data set is still available in the March data and, if it is, in which column it resides in the March spreadsheet. This brings to light the additional concern for the Noble Gas data that the documentation is not easily machine consumable, meaning that all mapping activities will need to be performed manually. Not only is this approach time consuming, but it also does not scale well into larger data sets.

The Copper data set was acquired during the process of a workshop to generate new methods of visualizing mineralogy data, initially on June 8, 2016. The process entailed trying various orders and organization for the data and results in various new versions of the data that depend on varying filtering requirements, acquired on August 21, 2016. Unlike the Noble Gas data set, the Copper Data had no accompanying documentation, since the primary consumers of the data at the workshop

were also mineralogy experts. However, this data set had more stable characteristics including physical and logical structure. Only two columns were removed from the transition to the second version, but sixteen new columns were added to the data collection. It also demonstrates a change in orientation with respect to data usage since the previous data set was designed to be distributed for general usage and discovery. In this case, the structure and organization of the data within the set was driven for a specific purpose in the development of more expressive visualizations. As a result, versioning information is driven by developmental needs instead of the other way around with versioning information bridging the gap between software migrations.

The data files from both data sets can easily be tracked using standard version management services such as GIT or SVN. Likewise, there exist comparison tools like Spreadsheet Compare from Microsoft Corporation that can generate diff-like outputs for each of the data sets. In conjunction with commit logs, the comparison outputs provide a basic versioning methodology that describes the data set's evolution. However, these applications rely on human attention and interaction to operate, and with larger data sets, proper documentation becomes difficult to maintain. With the Copper Data, the demand for new versions of the spreadsheets exceeded the time necessary to document version history as a result of rapid product evolution during the workshop. In consequence, the process to manually commit and annotate changed data impairs the natural progression of scientific development.

2.2 Database Systems

Databases remain the most relied upon technology for storing and searching large quantities of data rapidly. While the dynamic combination of tables means that data bases remain flexible enough to represent complex objects, it also means that they represent a much more complicated case for attribution. Since tables may be combined in different ways to answer complex queries, indexes do not remain constant across requests to the database. The approaches to database versioning typically focus on ensuring the reproducibility of queries to the database. This can often be difficult as with spreadsheets since changes to the content or structure can

result in different solutions from the database for the same query even using time stamps. For example, consider the query to select all columns of row A from a database on March 1st, then the database undergoes a schema change to add a new column to the table on April 1st. A subsequent request for all columns of row A would include the new column which does not represent the response on March 1st. In addition, even if the data is timestamped, the time signature is associated with the row and not the schema, meaning that the query may still return row A with the new column with a NULL value, depending on the distribution. The query, not the data, would need to be modified to exclude the new column.

This presents as a challenge because unlike data files and spreadsheets, databases are generally not instanced. Databases often store massive quantities of data and replication of that data to archive snapshots or distribution frequently proves too costly to be feasible. Instead, interaction with the database occurs from a centralized source through transactions. Various methods have been studied to manage changes within these systems focusing primarily on schema versioning, emphasizing data's structural component [77]. This provides a method to enact a transactional rollback on the database to execute queries in an environment reminiscent of the original execution. The framework of the resulting database environment can become quite complicated as a result of the complexity of the tables representing intricate data objects [78]. This results from the need to manage the time instances of realization, storage, and validity. The datum becomes realized at collection, then stored upon entry into the database, and finally valid until the present or new data replaces it. More recently, new methods have been developed to adjust to the enormous quantities of data populating modern databases, focusing on query citation rather than data citation [79] [80]. Citation by query avoids the complexities involved with referencing data that can grow and move. However, this method relies on the existence of a versioning system for data. This method also recognizes that modifying queries to operate on the current state of the database may often be easier than rolling back transactions or schema to reproduce the results of a query [81]. As a result, to versioning a database system may be more feasible as data size increases by applying methods to the query results and not to the data.

The RRUFF Database is "an integrated database of Raman spectra, X-ray diffraction and chemistry data for minerals" [82]. It features a web accessible change log using the transactional log generated by the database software[1]. As the records in specific tables change, the log reports these changes, supplying persistent access to the modifications made to the RRUFF data. The approach to this alteration information highlights the always on-line approach to modern databases where changes to the data do not constitute a new database. The log demonstrates strong versioning characteristics with not only a breakdown of the change components, but also a commentary on the motivation for the difference. In addition, its HTML structure allows automated web crawlers to systematically consume the version information. With the integration of web ontologies, the change log would also be intelligible to automated agents.

2.3 Ontologies

On-line ontologies are a different way of storing data than relational databases that has found significant traction within Semantic Web applications. They form graphs, relating a vocabulary of terms and relationships together to model complex interactions within an application's domain. Since the ontology is represented as a graph, it has more expressiveness than relational databases. The objects no longer need to share uniform structure and fields when entered into the database. Ontologies improve interoperability between scientific data sets by allowing differing data to share a common vocabulary and be comparable. Like other data, ontologies change regularly as definitions and relationships update to better represent their source material [83]. As a connected graph, they easily lend themselves to providing mappings between changes and versions within the ontology. New transitions would be represented as a simple link between new and old concepts. This is particularly important on the Semantic Web since most reasoning and interactions are handled automatically by underlying services. Ontologies, thus, benefit the most when providing both forward and backward mapping as it allows more up to date systems to interact with entities that haven't migrated yet [71]. Incomplete mappings, where

[1] `http://rruff.info/index.php/r=rruff_log_display`

transitions exclude either forward or exclude backward mappings, retain value as backward mappings inform traceability and forward mappings communicate advances in the domain. However, the uncertain landscape of web services means that full ontology mappings prove invaluable to making data inter-operable. Advances in ontology change detection have made tools which automatically generate mappings between versions of an ontology available [84]. However, in this project, the focus remains on improving the description of these mappings to provide not only descriptions but also explanations for the transition.

The Semantic Sea Ice Interoperability Initiative (SSIII) is a project combining the efforts of the National Snow and Ice Data Center (NSIDC) and the Rensselaer Polytechnic Institution (RPI) Tetherless World Constellation [85]. In their initiative, they developed an ontology to describe a core set of sea ice terms [86]. They have made available the second version of their ontology. The next edition of the ontology follows some adjusted best practices to improve modularity and coverage of the sea ice vocabulary. The new formulation would require the movement and addition of new concepts as well as the modification of other ontological entities. As of this writing, the ontology does not have a method to provide mappings between versions of the ontology even though the concepts are managed through GitHub. Currently, Ruth Duerr has developed a color coded visualization of the new ontology's layout using a CMap seen in Figure 2.4. The maps use a five color system with green representing no change, purple showing a moved concept, yellow coding an unmoved entity with changes to its name or definition, red showing a change to the ontology or nomenclature, and blue representing an entirely new concept.

The Global Change Master Directory (GCMD) is a metadata repository used by NASA to story records of its available data sets [87]. It employs a keyword ontology to search for Earth science data in NASA data sets. These keywords tag and label datasets into strictly defined categories in order to make them more discoverable [88]. Version 1.0.0 of the 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

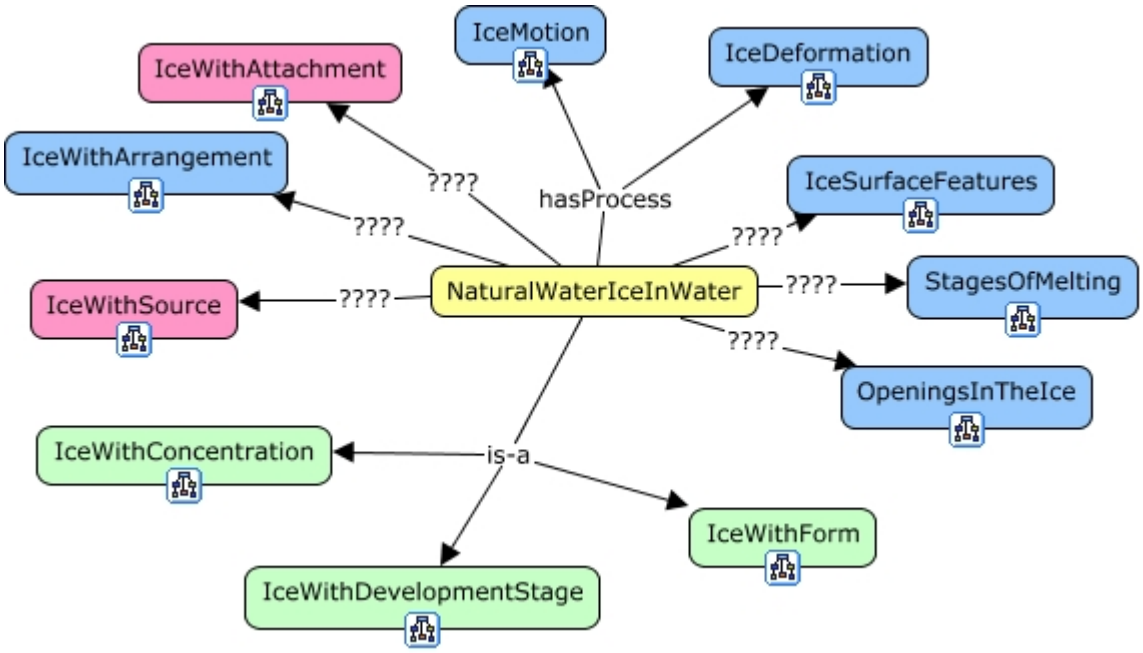


Figure 2.4: Concept map created by Ruth Duerr to organize the Sea Ice Ontology's Version 3 development.

represented a different level of the GCMD Keyword hierarchy: category, topic, and term, respectively. Incrementing a number in the version name indicates a change occurring in that level of the ontology. However, this gave very little information on compatibility between versions, and the ontology currently employs a more standard Major.minor release naming scheme.

The data set provides a very interesting case to study because of its history of medium storage. The GCMD originally distributed the keywords in a spreadsheet format, but later migrated onto database services as the scope and demand grew. The data may currently be accessed through a dynamic web service that can provide results in a variety of linked data formats. As a result, it leverages the endeavors made towards the environments mentioned in the previous two sections. The keywords have an accompanying change log, but due to the variety of mediums, the early logs are difficult to interpret. Since they attempt to use web technologies, the keywords each have unique identifiers that can be dereferenced using a Universal Resource Identifier (URI). Attribution, therefore, has mostly addressed by the source material. This is to be expected as a result of curated application of linked data

principles. Due to the shift from spreadsheet to databases, there exists a disconnect between the early versions of the ontology and modern editions. The work done in this project will be able to link them and provide a road map through the evolution and migration of the vocabulary as well as guide future evolutions of the keywords.

CHAPTER 3
DATASETS UTILIZED

3.1 Copper Dataset

The Paragenetic Mode for Copper Minerals database did not have these same constraints and also featured a more limited set of change. With the process's validity now verifiable, the versioning model could now apply to the resulting change log, but at that point, the model still included capturing the actual values in the data object that changed. Including the actual data into the change log gives concrete details as to how the object behaves when it changes, and is common practice. However, when modeling the version, data within the object provides a level of granularity that does not transport well from one information system to the next. In addition, the resulting linked data graph stores double the amount of data than the actual change, once for the linked data and again for the values. As a result, the model leaves out including the data. This allows the model to remain open and adapt to more complex versioning procedures.

3.2 Noble Gas Dataset

The "Noble gas isotopes in hydrocarbon gases, oils and related ground waters" database has the desirable qualities for this comparison of varied sizable provenance and multiple versions to provide comparable changes. However, gaps appeared to hinder the approach with using provenance data to measure change distance. To begin, each of the spreadsheet database's rows were considered to be a separate data object, as opposed to the individual file since this structure changes in the subsequent version, as explained later. Each row contained an entry indicating the reference used to compile the readings stored, and this entry was used as the data entity to produce a provenance graph with the PROV model as seen in Figure 5.1. An important challenge to note when creating these graphs is that in version 1, the references were stored in a very human readable fashion. The entry could be stored as a string or numeral even though all values were numbers. In addition,

the values were both comma separated and ranges indicated by a dash. Version 2 of the database corrects many of these problems with consistent content type and presentation. The documentation which accompanies the data set does not detail any changes to the compilation procedure that would indicate this improvement. The conclusion then follows that even though the two data objects have essentially the same provenance graph, it does not capture the operational change which has occurred within the data.

Version 2 also imposes many new changes that improve the data's readability. This can be seen with the unification from files per region to a single file, reduction of columns, and better standardization of value format within a column. The accompanying documentation includes instructions on how to read each column within the spreadsheet, but makes no mention as to the changes made to the original version to produce the current release. In software, this would take the form of a change log, but they also provide the developer a chance to explain his or her motivation for making those changes. In this case, an example would be the two versions reporting concentration in different units. As a result, the first goal to quantify the amount of modification between the first and second versions needed a change log document to codify the differences. For small applications, a listing of modifications sufficiently explains the transition to a new data set, but for larger applications a machine-readable change log demonstrates potential for significant value as previously mentioned.

Lack of familiarity with the data set and it's authors immediately posed a challenge to verifying the resulting change log's validity.

3.3 GCMD Keywords

GCMD Keywords do not qualify as a standard web ontology since it does not constitute a class hierarchy. As a result, the addition, removal, or modification of any term within the keywords has a significant impact on the semantics of using a keyword to describe a data set. More recent editions of the keywords, available through the Key Management Service (KMS), distributes they keywords in RDF format, and this allows them to be referenced through a unique web identifier. The

identifier drastically simplifies the attribution of changes between versions of the keyword list. However, older editions of were stored and distributed using Excel spreadsheets. This provides an interesting juxtaposition between versioning the spreadsheet as mentioned in a previous chapter and tracking the keyword changes. Results from this activity enables data sets described with different versions of the GCMD Keywords to remain discoverable within the same system.

Better quality data for the early versions of the keywords needs to be acquired before scripts can make mappings to newer releases. While easy to reference, the identifiers used to store the newer keywords are not immediately interpretable so further work needs to be done in order to form a mapping. Once the versions have been mapped, the workflow for publishing the change data follows the same process as in the previous chapter.

3.4 MBVL Classifications

CHAPTER 4
CONCEPTUAL MODEL

The goal of dataset versioning is to expose the relationships between versions of a dataset. To do this, the concept model relates three kinds of objects, versions, attributes, and changes, using three different orientations. The model obviously includes versions because they are the objects being compared. However, identifying one object as a version of another does not give much insight into the nature of their relationship. In order to characterize the change between two versions, the model relates their attributes. The model uses the Dublin Core Term hasPart to facilitate the link from a version to its attribute. The changes used then defines the difference between these attributes in each version. The modeling process can be viewed as creating a mapping between an original set and a new dataset. As mentioned previously, the operations conducted by data versioning systems boil down to primarily three operations: addition, invalidation, and modification. Since these activities are so prevalent, we use these three procedures to characterize the relationships between versions. A modification is straight forward to model because it maps together an attribute which exists in both versions, but addition and invalidation are a little different. Because the attribute doesn't exist in one version or in the other for addition and invalidation, it forms a '0 to 1' relationship between the attributes. This causes a problem conceptually because without a concept on one end, a mapping cannot be formed. The chosen solution was to use the version concept as the anchor in place of the non-existant attribute. This observation leads to the structure of the conceptual model used in this dissertation. The nature of change can be determined by observing the construction used to model a relationship and counting the number of attributes on each side. As a note, the figures in this chapter only depict the attribute relationships as 0 to 1, 1 to 0, and 1 to 1. That is, the cardinality of links entering a change concept from an attribute to the number of links exiting to an attribute. It is more valid to consider the relationships as 0 to X, X to 0, and X to Y in cardinality because it may take more than one attribute to identify an

observation in a version. For example, a cell in tabular data would need a row and a column to identify it, or a modification may change a single location attribute into two separate latitude and longitude entries.

An obvious concern about using these requirements to determine the mapping is that it does not guarantee meaningful versioning is being performed. In order to ensure that the relationships being exposed by the mapping is valid, we must go back to the definition of versions. Having common provenance establishes that performing a comparison between these two objects results in a relation pertaining to the same application. In addition, if the objects also can share the same workflow step it establishes that they have relatable content. This also addresses the possibility that we are comparing objects that have different purposes at separate points in a workflow, but share provenance as a result. Therefore, before applying the methods in this model, it must be first established that the two objects satisfy the requirements to be versions of each other.

4.1 MODIFICATION

The simplest operation to model is Modification because it has no missing parts. It maps a change from one attribute of version one to its corresponding attribute in version two. In Figure 4.1, a versioning comparison is being performed between two objects, Version 1 and Version 2. Each version has an attribute, Attribute 1 and Attribute 2, respectively. Finally, a change object connects the two attributes, denoting that the values described by the attribute are different.

Notice that the model captures neither the change's magnitude, nor the values of the attributes involved in the change. These are not included because too much domain knowledge will be required to interpret the significance of the value. In addition, the model would essentially begin storing a copy of the dataset, leading to space and redundancy concerns. If an application needs to include this information, they can be added as a property of the attributes involved, but this is outside the scope of this thesis. Simply knowing that the attribute has changed provides valuable information to identify the relationship between the two attributes from a versioning standpoint. Sometimes it may be necessary to distinguish between

modification changes. For this reason, the change concept in this relationship may be sub-classed to differentiate between different kinds of change that map one attribute to another. For example, in order to distinguish a modification in which the units of a measurement changes, a unit change concept, which sub-classes the modify change concept, can be used to connect the attributes from two different version.

The modification relationship is simple enough to understand, and a couple of well developed alternatives exist. Schema.org supplies a subclass of the UpdateAction labeled ReplaceAction. This concept has two properties, replacee and replacer which indicates that a new object replaces an old one. This provides a structure very similar to the one found in Figure 4.1 with the only difference being the relationship between Attribute 1 and Change M reversed. While this difference is subtle, it demonstrates a disagreement in view as to how change works in a system. Schema.org's perspective looks at a single replacement action in isolation and models that quite well. However, this versioning model views change as a quantity that flows through attributes from one version to another. Unlike ReplaceAction, ModifyChange does not attribute changes to an actor, but rather it acts as a medium to convey change between objects. PROV also has a property called isDerivedFrom to convey, "a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity," [9]. Changing this property into a qualified property then results in a form very similar to the construction in this model. This should not be very surprising since the inspiration for the current form comes from the way PROV uses qualified properties. It gains the advantage of turning the property into a concept that can be instanced and described easily using other ontologies. The primary discrepancy between this versioning ontology and PROV-O is that Version 2 no longer needs to depend on a previous entity. PROV assumes derivations to be part of a sequence, but situations exist, as seen in a later chapter, where research can create multiple simultaneous versions where an ordering does not matter.

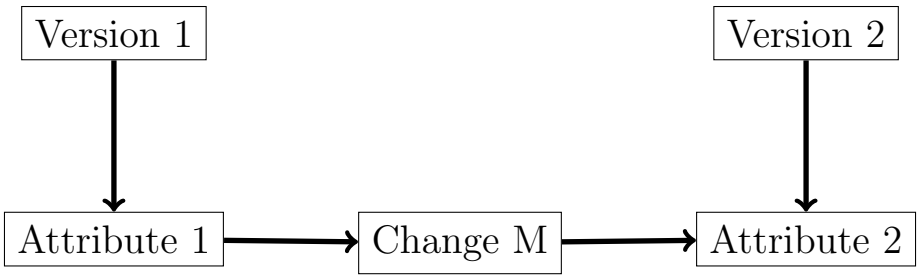


Figure 4.1: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

4.2 ADDITION

In Figure 4.2, the Addition model differs from the Modification construction by the absence of Attribute 1. This should mean that there does not exist a relationship between Version 1 and the concept Change A as the inclusion of Attribute 2 into Version 2 does not in general use content from the previous object. However, Change A in this model is not an activity, but a comparison relationship between Version 1 and 2. Therefore, a path must exist from Version 1 to Attribute 2. This formulation has the added benefit of conveying the idea of change as a quantity of flow, coming out of Version 1 and moving through the graph. In this context, the structure establishes the left-hand version object as a source of flow when attributes become added to the following object. This construction also forms a symmetric orientation with Invalidation.

In comparison, the AddAction from Schema.org considers addition as an activity performed by an agent. This view of addition does not take into account the prior state of the collection being manipulated. As a result, the proposed model for Addition should provide a better context for the relationships between objects when an attribute is added. The corresponding concept in PROV-O is a Generation. However, it relates together activities with entities, meaning that it could not relate together two versions or a version and an attribute. This property also does not make sense because, once again, no activity is producing the versioning relationship. Although, if we attribute new additions to the activity which generates Version 2, a relationship using Generation can be formed. This formation, however, would also apply to other attributes added to Version 2, and convolute the role the activity

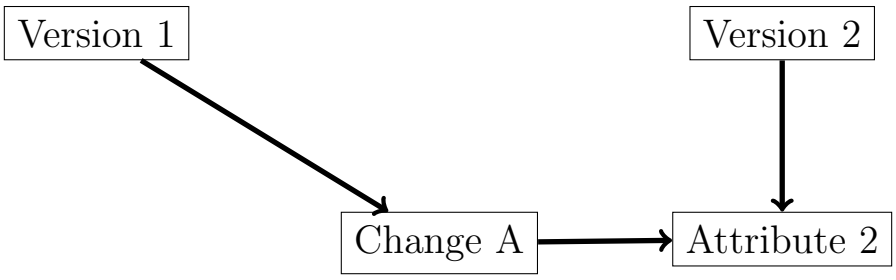


Figure 4.2: Model of the relationships between Versions 1 and 2 when adding an Attribute 2 to Version 2 as a result of Change A

took in an attributes generation. In the versioning model used here, each added attribute connects to a particular change instance so that they can be individually described.

4.3 INVALIDATION

The invalidation relationship has a missing attribute object on the other side of the relation, contrary to the addition construction. As a result of the invalidation, an attribute no longer exists in the following version. As seen in Figure 4.3, the invalidation change concept matches to the Version 2 object. Just like in addition model, this construction maintains a link between the two version objects. However, in this case, it makes more conceptual sense because Version 2 invalidates Attribute 1 by omitting it.

Related concepts include Schema.org's DeleteAction and PROV's Invalidation. PROV defines the property as the "start of the destruction, cessation, or expiry of an existing entity by an activity," but since nothing actively creates versioning relationships, this property is inappropriate for this application [9]. There is no activity to credit with removing the property in this comparison, and the absence results from a state-based property. Schema.org defines the DeleteAction as, "The act of editing a recipient by removing one of its objects," [**?**]. This definition assumes that an actor will perform an action upon an object or collection, removing one of its members, producing a new result. This follows the provenance mentality, but when versioning, the resulting object has already been created. Cases exist, as shown in Chapter 4, where an attribute is not included in a new version, thus indicating

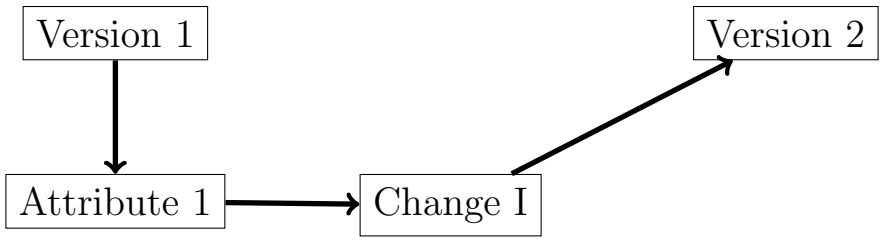


Figure 4.3: Model of the relationships between Versions 1 and 2 when invalidating Attribute 1 from Version 1 as a result of Change I

that it has been removed. However, the most important point of contention in this definition is the idea of removal and deletion. Although an object is no longer valid, it does not mean the data has been deleted. Invalid data often gets removed, but this is not always the case. As a result, naming the concept invalidation is more general and inclusive.

CHAPTER 5
VERSIONING TABULAR DATA

This chapter uses the Copper and Noble Gas databases to demonstrate the implementation of the model presented in the previous chapter.

5.1 Validate Comparisons

The first step in any versioning endeavor is to first establish that the two objects being compared meet the requirements of being each other's versions. For the copper dataset, the objects result from a compilation effort from the same sources of data. An entry recording the source material for each row in the table accompanies each reading. As a result, these entries were used to determine that they share similar provenance. They will also both be used as data inputs at the same step to generate data visualizations. The datasets become interchangeable within the context of executing the workflow. Determining whether the noble gas files are versions is a more challenging activity. From provenance graphs like the one constructed for the entry CAM001 in Figure 5.1, the entries share a common source document. Likewise, a significant portion of the other entries also share sources from the same body of work. The ones that do not are new or removed entries. As there was no personal involvement in the use of this data set, determining whether they can occupy the same workflow step requires actually looking into the files. Investigating column headers and the accompanying documentation reveals that they report many of the same measurements. Some amount of leeway is given in this assessment as having every quantity match would mean that the two files aren't versions, but the same object. This process helps to determine the confidence that using the model to encode the discovered changes will result in a meaningful versioning graph.

5.2 Form a Mapping

The next step involves formulating a method to determine whether a relationship constitutes an add, invalidate, or modify mapping. As discussed in the previous

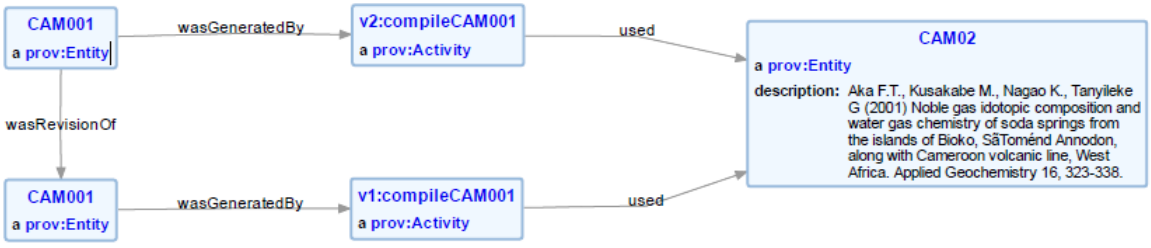


Figure 5.1: Provenance graph for the CAM001 entry of the Noble Gas Database. Other than the labels, the structure of each data object is very much the same.

chapter, this relies on determining whether an attribute exists in one version, the other, or both. However, with tabular data, a row and column attribute is required to match together specific cells. For the Noble Gas dataset, this is conveniently the entry identifier and the column headers. Row and column numbers are avoided because edits can result in the same entry associated with different indices. In addition, the first version of the Noble Gas dataset was organized into multiple files so the same row index would appear more than once, and therefore, could not be used to match together related entries. An observation that also simplifies identifying edits is that cells are rarely added or removed individually. When one row gains a cell all other rows gain one as well, perhaps with null values, forming a new column. As a result, data additions and invalidations only use one identifier. A basic method to determine the identifiers involved in an addition is straight forward. A set of identifiers $\mathcal{A} = \mathcal{R}_r - \mathcal{R}_l$ where $\mathcal{R}_l$ and $\mathcal{R}_r$ correspond to the row identifiers of the left-hand and right-hand versions, respectively. A converse procedure reveals the set of invalidated attributes $\mathcal{I} = \mathcal{R}_l - \mathcal{R}_r$. All the remaining identifiers exist in both versions and form a set of possible modification attributes. These are only possibilities because the mapping procedure does not check for differences, and as a result, rows that have not undergone a change are also included. The same is done with columns. This only performs a very basic version mapping, but data producers can significantly improve this with their more intimate understanding of the versions. For example, the base method would not understand that a Location column has been divided into two columns Latitude and Longitude. A data producer could manually link the Location attribute to the Latitude and Longitude

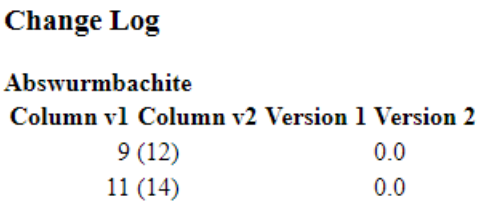
attributes, forming a single modification instead of an invalidate and two additions.

The copper minerals dataset starts each row with a unique mineral name, and these can be matched across datasets to determine if an entry has undergone a modification. The column headers have different formats because the initial file is an Excel file and the other was a comma separated file. As a result, the column headers were manually matched together with the remaining headers in each file mapped into the added and invalidated sets. The noble gas database also uses unique identifiers to mark each of its entries, but the first version of the database divides data among multiple files. For this reason, the identifiers must first be collected into a single dictionary data structure mapping identifiers to their files. The version map is then computed from this data structure. The database also uses multi-line headers which makes a basic automated approach difficult since the cells are not well aligned. This means that the columns for this database also had to be mapped manually to find the matching columns.

5.3 Produce Change Log

While a linked data versioning graph can now be generated using the mapping method, it is difficult to visualize without using more specialized software. The solution is to create a human readable change log to both validate the mapping matches and generate human readable change documentation. This addresses a gap in the Noble Gas data set's documentation which describes data use, but not data changes. The log is divided into three sections corresponding to the sets generated in Section 5.2. However, this provides an opportunity to address a weakness in modern change logs in that they are only human readable. Two technologies previously introduced, RDFa and JSON-LD, provide a means of embedding linked data into HTML documents, allowing the log to be both human and machine readable.

Consider the entry in Figure 5.2 from a change log of the Copper data set. RDFa uses the attributes in HTML documents to include linked data describing content on the page. This means that an automated web agent can read the document and extract the associated versioning graph. A selection of the marked up source from Figure 5.2 appears in Listing 5.1. Only the lines displaying table head-



Change Log

Abswurmbachite

Column v1	Column v2	Version 1	Version 2
9 (12)		0.0	
11 (14)		0.0	

Figure 5.2: Abswurmbachite entry in the Copper Dataset Change Log

ers and the second data line have been removed. Immediately, it can be seen that the markup is quite cluttered as a result of the clash between RDFa's intent and its use. The goal of this technology is to describe the content on a page and give context to particular values, marking a string of digits as a phone number for example. However, in this application, very little of the content is used in the model except for line 3, which uses the mineral name as an attribute label. It is instead used to encode the versioning model into the HTML document's lattice, leveraging the ability to imply relationships in nested tags. Unfortunately, RDFa interpreters are very particular in the way they resolve implications, and the order in which data appears in a change log to be human readable does not match the order they need to be for RDFa to encode the model. In lines 10 and 11, two triples have to be explicitly included in order to conform with the model outlined in Chapter 4.

```
1 <h3>Change Log</h3>
2 <div about="Version1" rel="vo:hasAttribute">
3   <div resource="v2:Abswurmbachite" typeof="vo:Attribute">
4     <span style="font-weight:bold" property="http://www.w3.org
        /2000/01/rdf-schema#label">Abswurmbachite</span>
5     <table rel="vo:Undergoes">
6       <tr about="ChangeAbswurmbachite12" typeof="vo:Change">
7         <td align="right" rev="vo:Undergoes" resource="v1:
            AttributeAbswurmbachite12v1" typeof="vo:Attribute"> 9</td>
8         <td property="vo:resultsIn" resource="v2:
            AttributeAbswurmbachite12v2" typeof="vo:Attribute">(12)</
            td>
9         <td> </td>
```

```
10        <td> 0.0</td>
11        <span about="Version1" property="vo:hasAttribute" resource="
              v1:AttributeAbswurmbachite12v1"></span>
12        <span about="Version2" property="vo:hasAttribute" resource="
              v2:AttributeAbswurmbachite12v2"></span>
13      </tr>
14    </table></div></div><br>
```

Listing 5.1: Abswurmbachite RDFa

The difficulties adopting RDFa into the change log arise from a misalignment in intent and use. Alternatively, a technology meant to store data in web documents, such as JSON-LD, may produce better results. Listing 5.2 provides the alternative encoding of the Abswurmbachite entry from RDFa. While significantly longer, the number of triples is not limited by the tag count within a change. It also separates the human-readable content from the model data, making the source easier to code and modify. Additionally, a decision was made to divide each change's linked data into it's associated row instead of collecting them all at the bottom or top of the page in a single script node. The practice of a one node collection is generally helpful for many web applications to load data quickly, but since this is not an application, it makes more sense to break the content up. Changes to individual attributes can be identified using anchors on the web page, then agents need only extract and parse the link data to these specific entries. This way, a subgraph of only the pertinent attributes can be created without first ingesting the entire versioning graph. However, a problem both of these change logs struggle with is size. The Copper Minerals data set encoded in RDFa is 1.7 MB and JSON-LD is 3.3 MB. In the Noble Gas data set, the RDFa and JSON-LD change log sizes are 59 and 60 MB, respectively. The Noble Gas change logs often do not load in a browser.

```
1 <h3>Change Log</h3>
2 <div about="v1:Abswurmbachite">
3   <span style="font-weight:bold" property="http://www.w3.org/2000/01/
        rdf-schema#label">Abswurmbachite</span>
```

```
4   <table>
5     <tr id="ModifyChangeAbswurmbachite12">
6       <td align="right"> 9</td>
7       <td >(12)</td>
8       <td> </td>
9       <td> 0.0</td>
10      <script type="application/ld+json">
11 [
12 {
13        "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.
             jsonld",
14        "@id": "http://CUdb.com/v1/AttributeAbswurmbachite9",
15        "@reverse": {
16               "hasAttribute": "Version1"
17        },
18        "@type": "vo:Attribute",
19        "label": "Primary",
20        "undergoes": "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/
             CUjsonlog.html#ModifyChangeAbswurmbachite12"
21 },
22 {
23        "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.
             jsonld",
24        "@id": "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/
             CUjsonlog.html#ModifyChangeAbswurmbachite12",
25        "@type": "vo:ModifyChange",
26        "resultsIn": "http://CUdb.com/v2/AttributeAbswurmbachite12"
27 },
28 {
29        "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.
             jsonld",
```

```
30        "@id": "http://CUdb.com/v2/AttributeAbswurmbachite12",
31        "@reverse": {
32                "hasAttribute": "Version2"
33        },
34        "@type": "vo:Attribute",
35        "label": "Primary"
36 }
37 ]
38      </script>
39    </tr>
40  </table></div><br>
```

Listing 5.2: Abswurmbachite JSON-LD

5.4 Generate Versioning Graph

A versioning graph is now generated using the mapping method determined in Section 5.2. Since the attributes in sets $\mathcal{A}$ and $\mathcal{I}$ guarantee a change, each item is encoded into linked data and output to a document.

```
1 <http://rdfa.info/play/Version1> a vo:Version ;
2        vo:absentFrom <http://rdfa.info/play/AddChange21> .
3 <http://rdfa.info/play/AddChange21> a <https://orion.tw.rpi.edu/~blee
    /VersionOntology.owl#AddChange> ;
4        vo:resultsIn <http://rdfa.info/play/Attribute21> .
5 <http://rdfa.info/play/Attribute21> a <https://orion.tw.rpi.edu/~blee
    /VersionOntology.owl#Attribute> ;
6        rdfs:label "EGY001"
7 <http://rdfa.info/play/Version2> a vo:Version ;
8        vo:hasAttribute <http://rdfa.info/play/Attribute21>
```

Listing 5.3: Noble Gas Add in Turtle

Listing 5.3 presents the statements in turtle format necessary to express that the

entry EGY001 has been added to the dataset from Version 1 to Version 2 as shown in Figure 5.3. Notice that the namespace for many of the URIs is ⟨http://rdfa.info/play/⟩. This results from the triples being extracted out of an HTML change log with embedded linked data, and this is used as the default namespace for the page. Since we know the number of additions, each change instance can be easily numbered. Add changes are also kept separate to allow for individual annotation. Likewise, the graph generator iterates over the set of invalidations and generates a set of statements to instantiate each invalidate change.

However, Figure 5.3 also demonstrates an interesting set of decisions made early in the generation process regarding modifications. Firstly, the relation presented in the figure is unbalanced and the right-hand side of ChangeCAM00111 links only to the column identifier but not to the corresponding row attribute. This links from a mismatch between the model's structure, the order in which data appears in the change log, and the way RDFa links properties together. Because the row label forms the outermost encapsulation, it cannot instantiate both row identifiers and implicitly link them separately. To do so would require explicitly instantiating the attribute in a non-visible part of the document which would defeat the purpose of using RDFa to implicitly encode the versioning graph into the document. Secondly, the column identifier AttributeCAM00111v1 combines together the row label and the column number, making it unique to just the CAM001 row. This was a decision to properly identify the attribute as it appears in the change log document. Since the item is not a generic column 11 element in the log, but a specific item of CAM001, the entry id is included in the column identifier. This formulation poses a problem when trying to query the graph and determine how many rows have a column 11 modify change for example. When the versioning triples are not extracted from a change log and printed as linked data, the structure has greater freedom as seen in Figure 5.5.

5.5 Multiple Linked Versions

The figures in this document so far have depicted a comparison between only two versions, but versioning often involves more than two objects, either in sequence

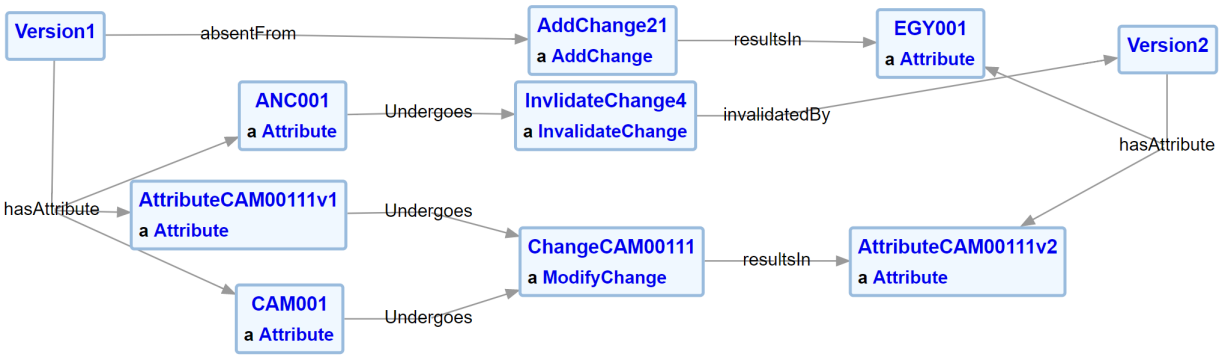


Figure 5.3: Some initial entries from versions 1 and 2 of the Noble Gas dataset

or parallel. Figure 5.5 shows a graph that follows change as it moves through three versions of the Noble Gas data set. From the first to second version of the data, EGY001 becomes introduced as an attribute into the dataset. This entry then undergoes a modification change in columns 29, 31, and 43 when comparing versions two and three. The construction results in a format naturally oriented to show the flow of change from version one to three. The advantage of this formation is that now many versions can be related together using a unified set of semantics that cannot be achieved through combining the other concepts and properties in Chapter 4. Versioning forms a continuous relationship even into later versions without introducing new semantics. This is important since many versioning linked data alternatives view version change as a single contained activity.

When considering multiple versions, a particular challenge is when an attribute does not change. In that particular case, no links are created, but if this occurs between two transitions, the flow of change is broken. For example, in Figure 5.5, column 31 of EGY001 becomes modified transitioning into the third version. If that column underwent no activity in the next transition but changed from version four to five, the connection between all the column 31s would no longer be continuous. This poses a problem for executing queries in a triple store which rely on graph traversals, but no path exists between the two modification changes in the example.

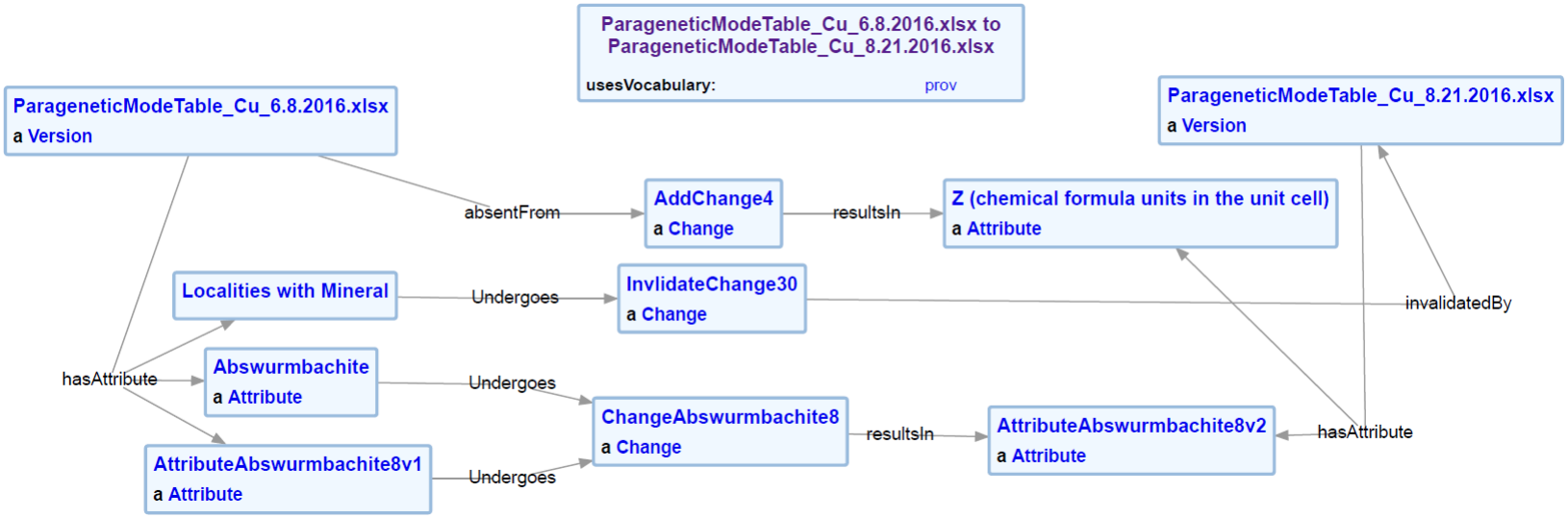


Figure 5.4: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications.

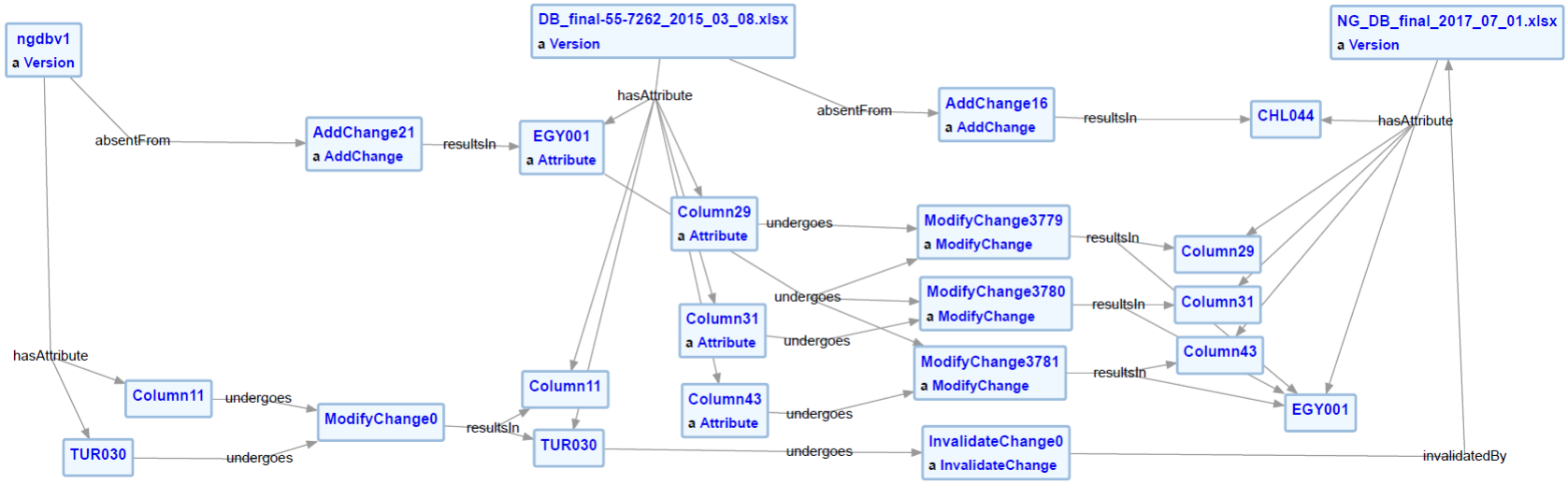


Figure 5.5: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications after the publication of the third version.

CHAPTER 6
ONTOLOGY AND TAXONOMY VERSIONING

This chapter extensively uses the GCMD Keywords data set to study methods of determining the extent of change in a data set. The keywords have a long sequence of versions, allowing trends to be observed over the course of releases.

6.1 Creating the Versioning Graph

The Global Change Master Directory maintains and releases the different versions of their keyword list. From this, it can be concluded that each edition shares provenance and can be used in the same workflow step. This conclusion is further justified as each keyword concept uses the same Unique Resource Identifier (URI) across versions. The identifiers also act as an ideal key in the version mapping. While additions and invalidations are simple to identify using these keys, modifications are not since a change in their key would result in completely different object. Instead, the immediately broader concept is looked at. Each keyword uses the concepts skos:Broader and skos:Narrower, where skos refers to the Simple Knowledge Organization System ontology name space, to form a tree hierarchy with the broadest concept "Science Keywords" forming the root. A modification would then result if a concept moved to a different place in the hierarchy. This would result in the removal of a child node from the parent and a different broader concept for the child, meaning two modifications occur. However, in this project, only the child is recorded since it is the concept that moves around in the hierarchy. Versioning graphs for each comparison was generated by extracting JSON-LD from the corresponding change log, and entering the triples into a Fuseki triple store.

6.2 Quantifying Change

The GCMD group migrated their keywords into a centralized Keyword Management System (KMS) as of June 12, 2012. Each subsequent keyword release has been supplied an identifier by the management group and the add, invalidate, and

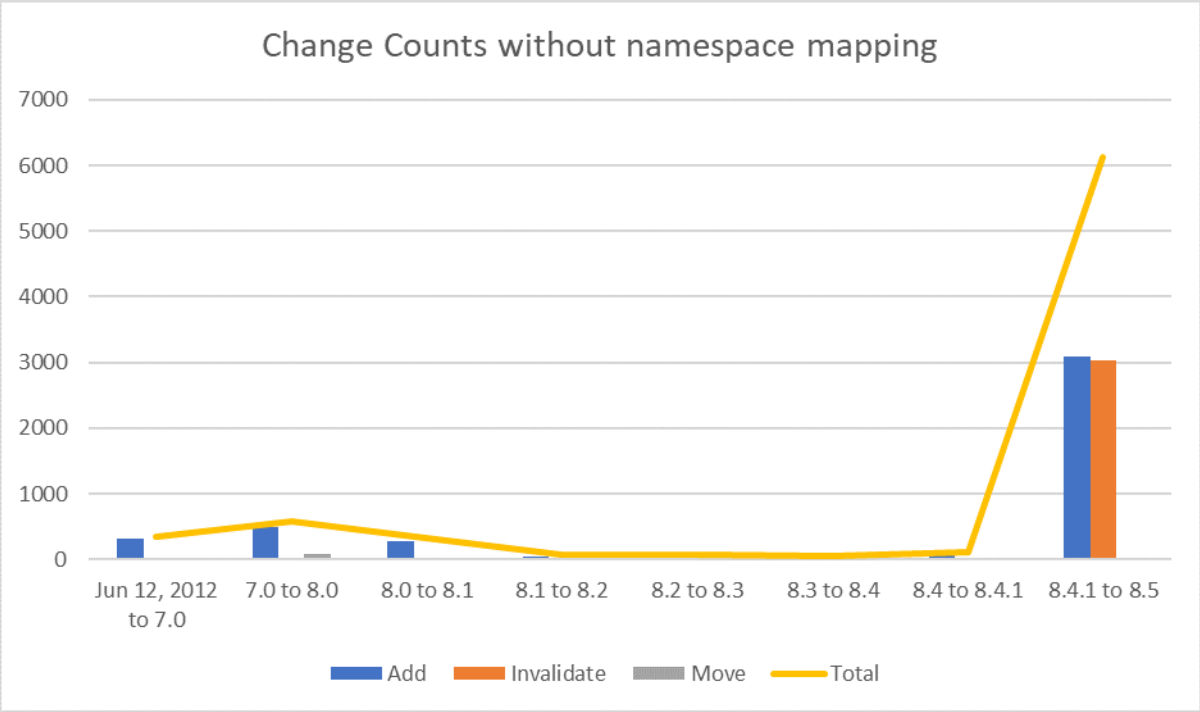


Figure 6.1: Add, Invalidate, and Modify counts in Version 8.5. The counts show change magnitudes and indicate that major and minor changes differ by orders of magnitude.

modify counts between each transition are presented in Figure 6.1. The query used to extract the counts is found in Listing 6.1. Notice the sharp spike in adds and invalidates when transitioning from version 8.4.1 to 8.5. Not only should a small transition not produce changes of this magnitude, but the data sets size is on the order of the recorded invalidates. In addition, no modifications are revealed, and even the root node "Science Keywords" has been invalidated. Further investigation of the root word reveals that the name space for the keywords has changed from HTTP to HTTPS. Since the identifiers are unique, this means they no longer refer to the same object after the protocol change. This results in the whole data set being invalidated and a new data set being added. However, the dot decimal identifier only indicates a minor change, demonstrating a difference between the producer's perceived divergence and the actual change. To provide context, NASA mandated a transition to secure protocols, and the group changed the named space to ensure the URIs remained resolvable.

```
1 PREFIX vo:<http://orion.tw.rpi.edu/~blee/VersionOntology.owl>
2 PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
3
4 SELECT ?p (COUNT (DISTINCT ?s) as ?count)
5 {
6        ?s a ?p .
7        ?p rdfs:subClassOf vo:Change .
8 } GROUP BY ?p
```

Listing 6.1: This query compiles the counts for each subclass of Change in a GCMD versioning graph

That the data producers did not perceive this change in name space to be a major modification can be demonstrated by accounting for the change and recounting. In the modified mapping, HTTP and HTTPS identifiers are treated the same. Differences in change magnitudes become much clearer after controlling for the altered name space in Figure 6.2. All revisions are dominated by additions, but major version changes have counts around 300 to 500 while minor revisions are an order of magnitude smaller. This includes the transition from version 8.4.1 to 8.5. From the identifier scheme and the change counts, it is clear that the keyword management team expected only minor changes in the keywords. This analysis demonstrates that relying on data producers to name their versions using the dot decimal identifiers based on their perceived change also relies on their perceiving the intended utilization of their data set by all their users. The count results seem to indicate that they can differentiate between major and minor revisions, but it also shows that current version labels may not capture all the change within a transition.

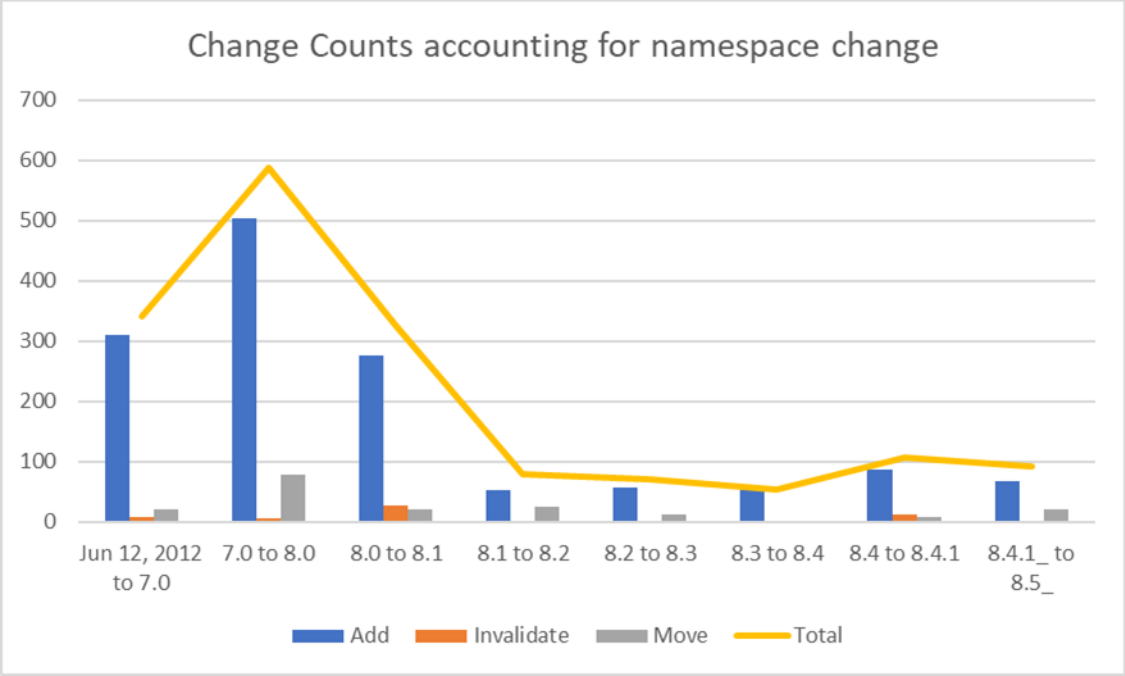


Figure 6.2: Add, Invalidate, and Modify counts ignoring the namespace changes in Version 8.5. The counts show change magnitudes appropriate for the identifier.

CHAPTER 7
MBVL CLASSIFICATION

The goal of this section is to use the versioning graph to compare the accuracy of different algorithm and taxonomy combinations in determining the taxonomic classification of marine microbiological species.

7.1 Versioning Graph

The experiment undergoes two phases of comparison in this procedure. The first phase compares the initial species content with the classifications by a particular algorithm/taxonomy combination. Since the classification results from a population selected according to the initial species list, these two datasets share a common provenance. However, the list cannot be used in place of the classifier results because not only does it have only 21 entries, but also it does not have a clear correspondence with the DNA chains sent to the classifier. As a result, these two sets are no versions of each other, and versioning results will have weak implications. A labeling of the initial input data, alternatively would be considered a version of the results, but that data product is not available.

Each taxonomy and classifier combination outputs a taxonomic classification for each entry from the same source. Shared input data indicates the results share common provenance. The classifications also share the same workflow step because their results have similar formats, reporting a specific taxonomic name for each entry. Since classifications occur over the same set of entries, their identifiers can be used to match outputs together for comparison. However, if this method of matching is used, every mapping would be a modification since each identifier appears in all data sets, and it would not provide any comparison based on the algorithm's specificity. Instead, a mapping using the accuracy of each algorithm is used. Since a name is assigned at a taxonomic rank to a sequence only if it passes the algorithm's confidence level, matches can be determined on whether a classifier can confidently decide more or fewer ranks. As a result, additions and invalidations based on whether a classifier

can ascertain more or less of an entry's taxonomic name while modifications indicate the same specificity but mismatching names. This method of mapping the versions together allows the results to give insight into the accuracy of different algorithm and taxonomy combinations.

7.2 Analysis

Figure 7.1 shows the results of four comparisons performed using the matching procedure in the previous section. There are only four comparisons because varying both taxonomy and algorithm muddles the contribution of each towards a more accurate result. In the first set of columns, the Silva taxonomy results are versioned against RDP using the Spingo algorithm. The naming reflects the orientation in the versioning graph so Silva forms the left-hand version and RDP would be the right-hand version. In this comparison, using the RDP taxonomy seems to provide more accurate results, most specifically at the species level. The taxonomies also disagree fairly often at the species and family ranks. Switching to the Gast algorithm in the second set of columns, RDP once again demonstrates a noticeably greater accuracy in species classification. There is also significantly fewer disagreements using the Gast algorithm between the two taxonomies. Looking at the third set of columns, Silva demonstrates greater accuracy classifications under the Spingo algorithm than under Gast. Over four thousand of these entries can be classified to the species level when Gast cannot. In the fourth set of columns, RDP appears to perform better with Spingo than Gast. However, the comparison is dominated by a much larger number of disagreements between almost six thousand entries, primarily at the species rank. On closer inspection, this disagreement is explained by Gast classifying the species for a number of entries as unclutured bacterium. This analysis presents evidence that using the RDP taxonomy with the Spingo algorithm will produce the most accurate classification results.

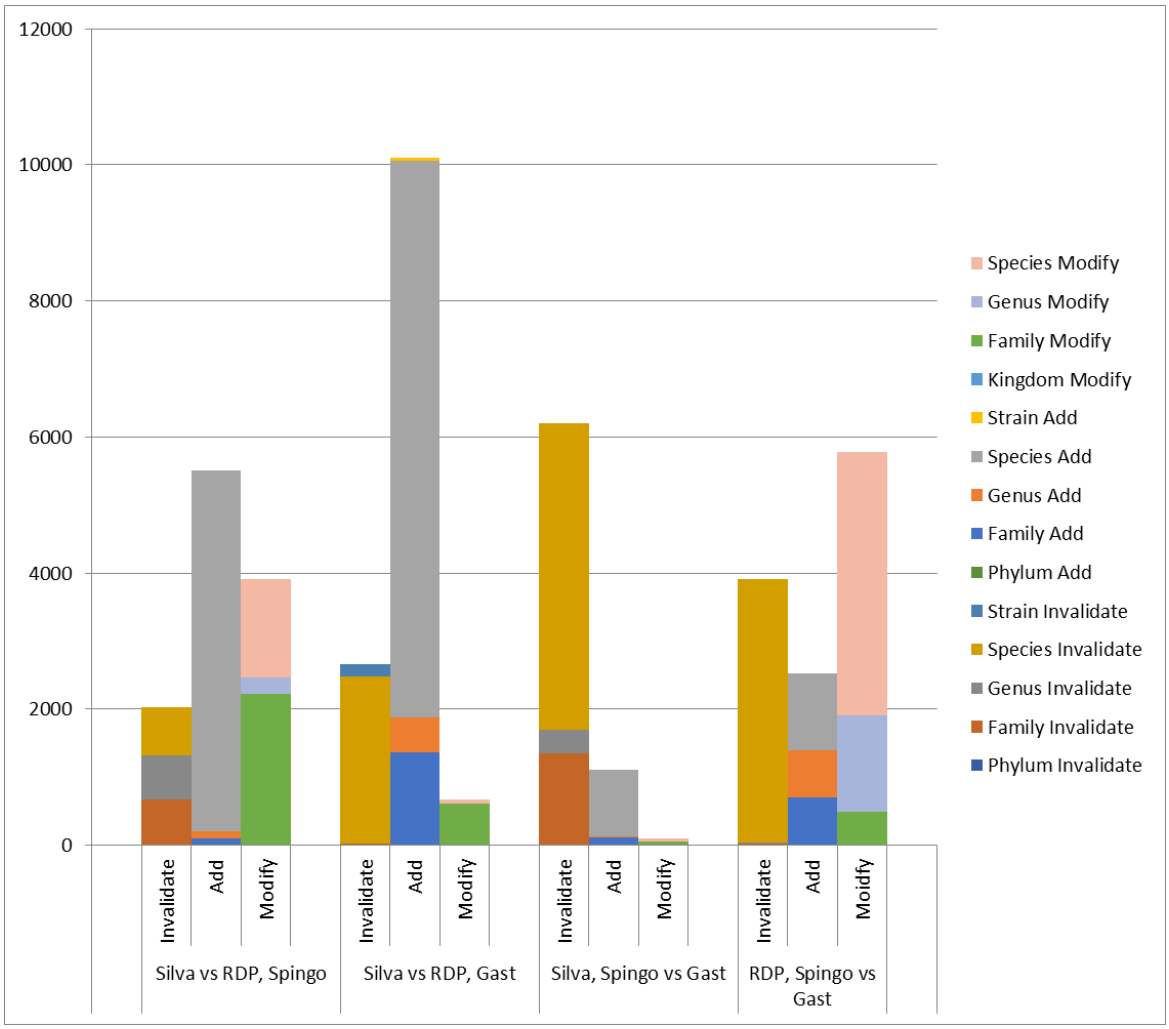


Figure 7.1: Compiled counts of adds, invalidates, and modifies grouped by taxonomic rank across algorithm and taxonomy combinations.

CHAPTER 8
DISCUSSION

8.1 Model

The resulting model addresses versioning by looking at the attributes of each version. Other ontologies takes a higher level view in terms of version modeling. While it is more specific, this implementation forces some space requirements. PROV only requires 3 to 5 triples in order to make a versioning statement. This model uses 9 triples for a mod change and 7 to encode addition and invalidation. To model a version has space complexity of $O(7M + 5(A + I))$ since the version declaration statements overlap. However, a similar structure can be achieved using prov:wasDerivedFrom to replace modifications and schema:AddAction and schema:DeleteAction to replace additions and invalidations. The resulting space complexity is $O(7M + 3A + 5I)$. This is fairly similar with additions seeing a reduction since the left-hand version no longer contributes to the AddAction. Thus the primary benefit of using this model comes from semantics.

The reason prov:Generation and prov:Invalidation are not used is because they expect an activity to be responsible for an object. However, it is not generally true that an action actively added or removed an attribute from an object in the left-hand version to produce the right-hand revision. That mentality denies the ability to conduct versioning comparisons between objects that are not sequentially adjacent. The activity producing a far away object may not immediately relate to the original data in a version comparison, resulting in a situation where it would be inappropriate to use the two PROV concepts. When considering versioning in a state-based sense, relationships exist as a result of two objects being versions of each other.

8.2 Implementation

The versioning process breaks down to three formal steps which appear in all contexts of versioning studied in this thesis. The first activity verifies that the ob-

jects being compared are actually versions of each other. This exercise is often left out of details in practice since a data producer is often fairly certain as to the state of their versions. Mechanisms are otherwise employed to enforce a strict documentation procedure to ensure the data's comparability as seen in version control software. However, this step establishes the foundation and validity of further actions taken to version the objects. This ensures that a mapping can be performed and will be meaningful. The next step is generating the mapping to identify addition, invalidation, and modification relationships. The resultant mapping in spreadsheet comparisons followed very similar rules, but when looking at the MBVL dataset, the definitions were changed to achieve a specific goal. The final step involves publishing the change information using the mapping. In this thesis, the resulting product is published into a versioning graph.

One of the desired contributions was to study the possibility of a machine readable change log. In this implementation, the number of triples necessary to implement the model significantly impairs the log's ability to remain human readable. The Noble Gas data set's change logs could not be loaded using a web browser. One contributor to this problem is that the modification of an entire column would result in multiple entries equal to the number of rows in the table. These entries could be combined together into a single statement relating just the effected columns. However, this optimization would greatly impact the resulting change counts, reinforcing that version analysis depends largely on the mapping method, but this would likely allow the log to become readable. JSON-LD proves to be a better mechanism for encoding the versioning graph than RDFa since it is intended to encode data while the latter primarily contextualizes visible content.

When linking together multiple versions using a versioning graph, the relationship between non-adjacent editions remains implied in the graph's structure. The natural pathway between attributes in non-adjacent versions holistically considers the relationships among all attributes along that path. In comparison, other models only capture activity between the adjacent versions.

8.3 Version Identification

The versioning process discovered a discrepancy in the identifier assignment in the GCMD Keywords taxonomy. The original analysis was intended to determine if dot-decimal identifiers could be predicted using the change counts of the versioning graph. However, version 8.5 was named with respect to perceived taxonomy changes and did not consider underlying linked data practice revisions. This brings into question the accuracy of all prior names and the any relationships observed between identifier and change counts. It would explain how 8.4.1 had more additions than any previous minor change but obtains a third bracket identifier However, assuming that the observed relationship remains, it can be shown that the keyword management team did not consider the namespace change as a major modification in their data set. This is seen after accounting for namespace differences to show that the change count magnitude resembles other versions in the same identifier bracket. This brings into concern the practice of version name assignment based on producer perception and not on more concrete measures. A poor understanding in the amount of change between two versions can lead to flawed expectations in migrating across them.

This is not to claim that change magnitudes should be the sole mechanism in determining version identifiers. However, it can provide a more quantitative characterization of changes within the system. In Figure 6.2, the yellow line indicates the total changes made to the data set, performing a similar function as the major/minor/revision version identifier. However, breaking up the changes into types reveals the dominant contribution of additions to the data set. This understanding of the data's behavior cannot be revealed with a three number dot decimal identifier system. However, it also does not take into account the possibility that single or small changes can have significant implications scientifically.

8.4 Change Analysis

In Chapter 7, the versioning process was used to compare the performance of different taxonomy and algorithm combinations. This diverges from many of the common understandings of derivations since each of the versions are not sequential and are largely independent. The application demonstrated a case in which

two data sets do not form versions. Since they are not in a state of being revisions, implementing the version model on these two data sets would produce largely meaningless relations. The criteria of determining valid states may seem rather dubious, and there are likely other criteria which decide more conclusively. The data sets in this work, however, only reveal these two requirements to justify the use of versioning relationships.

Versioning models often provide documentation on changes between versions so it is interesting applying it actively to perform analytics. The results are able to provide a multi-lateral characterization of differences between modifications to either the choice in taxonomy or algorithm. In order to achieve more specificity, each of the core operation concepts had to be sub-classed in order to also consider taxonomic rank.

REFERENCES

[1] M. Branco, D. Cameron, B. Gaidioz, V. Garonne, B. Koblitz, M. Lassnig, R. Rocha, P. Salgado, and T. Wenaus, "Managing atlas data on a petabyte-scale with dq2," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062017, 2008. [Online]. Available: http://stacks.iop.org/1742-6596/119/i=6/a=062017

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

[3] R. E. Duerr, R. R. Downs, C. Tilmes, B. Barkstrom, W. C. Lenhardt, J. Glassy, L. E. Bermudez, and P. Slaughter, "On the utility of identification schemes for digital earth science data: an assessment and recommendations," *Earth Science Informatics*, vol. 4, no. 3, p. 139, 2011. [Online]. Available: http://dx.doi.org/10.1007/s12145-011-0083-6

[4] M. Dummontier, A. J. G. Gray, and M. S. Marshall, "The hcls community profile: Describing datadata, vversion, and distributions," in *Smart Descriptions & Smarter Vocabularies*, 2016. [Online]. Available: https://www.w3.org/2016/11/sdsvoc/SDSVoc16_paper_3

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

[6] B. R. Barkstrom, *Data Product Configuration Management and Versioning in Large-Scale Production of Satellite Scientific Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 118–133. [Online]. Available: http://dx.doi.org/10.1007/3-540-39195-9_9

[7] "Rdfa 1.1 primer - third edition: Rich structured data markup for web documents," March 2015, accessed: December 24, 2016. [Online]. Available: http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/

[8] W. Goddard and H. C. Swart, "Distances between graphs under edge operations," *Discrete Math.*, vol. 161, no. 1-3, pp. 121–132, Dec. 1996. [Online]. Available: http://dx.doi.org/10.1016/0012-365X(95)00073-6

[9] T. Lebo, S. Sahoo, and D. McGuinness, "Prov-o: The prov ontology," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-o-20130430/

[10] C. Tilmes, Y. Yesha, and M. Halem, "Distinguishing provenance equivalence of earth science data," *Procedia Computer Science*, vol. 4, pp. 548 – 557, 2011.

[Online]. Available:
http://www.sciencedirect.com/science/article/pii/S1877050911001153

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

[12] B. Barkstrom, *Earth Science Data Management Handbook: Users and User Access*. CRC Press, April 2014, vol. 1. [Online]. Available:
https://books.google.com/books?id=pI3rTgEACAAJ

[13] S. Burrows, "A review of electronic journal acquisition, management, and use in health sciences libraries," *Journal of the Medical Library Association*, vol. 94, no. 1, pp. 67–74, 01 2006, copyright - Copyright Medical Library Association Jan 2006; Document feature - Graphs; Tables; ; Last updated - 2016-11-09. [Online]. Available:
http://search.proquest.com/docview/203517273?accountid=28525

[14] K. Berberich, S. Bedathur, T. Neumann, and G. Weikum, "A time machine for text search," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '07. New York, NY, USA: ACM, 2007, pp. 519–526. [Online]. Available: http://doi.acm.org/10.1145/1277741.1277831

[15] S. Lyons, "Persistent identification of electronic documents and the future of footnotes," *Law Library Journal*, vol. 97, pp. 681–694, 2005.

[16] B. R. Barkstrom, T. H. Hinke, S. Gavali, W. Smith, W. J. Seufzer, C. Hu, and D. E. Cordner, "Distributed generation of nasa earth science data products," *Journal of Grid Computing*, vol. 1, no. 2, pp. 101–116, 2003. [Online]. Available: http://dx.doi.org/10.1023/B:GRID.0000024069.33399.ee

[17] P. Ciccarese, S. Soiland-Reyes, K. Belhajjame, A. J. Gray, C. Goble, and T. Clark, "Pav ontology: provenance, authoring and versioning," *Journal of Biomedical Semantics*, vol. 4, no. 1, p. 37, 2013. [Online]. Available:
http://dx.doi.org/10.1186/2041-1480-4-37

[18] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo, "Version management of xml documents," in *Selected Papers from the Third International Workshop WebDB 2000 on The World Wide Web and Databases*. London, UK, UK: Springer-Verlag, 2001, pp. 184–200. [Online]. Available:
http://dl.acm.org/citation.cfm?id=646544.696357

[19] M. Macduff, B. Lee, and S. Beus, "Versioning complex data," in *2014 IEEE International Congress on Big Data*, June 2014, pp. 788–791.

[20] A. Stuckenholz, "Component evolution and versioning state of the art," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 1, pp. 7–, Jan. 2005. [Online]. Available: http://doi.acm.org/10.1145/1039174.1039197

[21] "Common questions: Ubuntu release and version numbers," Canonical Ltd., accessed: December 12, 2016. [Online]. Available: https://help.ubuntu.com/community/CommonQuestions##Ubuntu%20Releases%20and%20Version%20Numbers

[22] 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

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

[24] S. Payette and T. Staples, *The Mellon Fedora Project Digital Library Architecture Meets XML and Web Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 406–421. [Online]. Available: http://dx.doi.org/10.1007/3-540-45747-X_30

[25] K. S. Baker and L. Yarmey, "Data stewardship: Environmental data curation and a web-of-repositories," *The International Journal of Data Curation*, vol. 4, no. 2, pp. 12–27, 2009.

[26] J. Kovse and T. Härder, "V-grid-a versioning services framework for the grid," in *Berliner XML Tage*, 2003.

[27] K. Holtman, "CMS Data Grid System Overview and Requirements," CERN, Geneva, Tech. Rep. CMS-NOTE-2001-037, Jul 2001. [Online]. Available: http://cds.cern.ch/record/687353

[28] R. Rantzau, C. Constantinescu, U. Heinkel, and H. Meinecke, "Champagne: Data change propagation for heterogeneous information systems," in *In: Proceedings of the International Conference on Very Large Databases (VLDB), Demonstration Paper, Hong Kong*, 2002.

[29] B. Tagger, "A literature review for the problem of biological data versioning," Online, July 2005. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/btagger/LitReview.pdf

[30] U. K. Wiil and D. L. Hicks, "Requirements for development of hypermedia technology for a digital library supporting scholarly work," in *Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 2*, ser. SAC '00. New York, NY, USA: ACM, 2000, pp. 607–609. [Online]. Available: http://doi.acm.org/10.1145/338407.338517

[31] D. Dai, Y. Chen, D. Kimpe, and R. Ross, "Provenance-based object storage prediction scheme for scientific big data applications," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 271–280.

[32] P. Vassiliadis, M. Bouzeghoub, and C. Quix, *Towards Quality-Oriented Data Warehouse Usage and Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 164–179. [Online]. Available: http://dx.doi.org/10.1007/3-540-48738-7_13

[33] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: A survey," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 1–28, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1057977.1057978

[34] 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

[35] R. Cavanaugh, G. Graham, and M. Wilde, "Satisfying the tax collector: Using data provenance as a way to audit data analyses in high energy physics," in *Workshop on Data Lineage and Provenance*, Oct. 2002.

[36] P. P. da Silva, D. L. McGuinness, and R. Fikes, "A proof markup language for semantic web services," *Information Systems*, vol. 31, no. 45, pp. 381 – 395, 2006, the Semantic Web and Web Services. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437905000281

[37] M. Bouzeghoub and V. Peralta, "A framework for analysis of data freshness," in *Proceedings of the 2004 International Workshop on Information Quality in Information Systems*, ser. IQIS '04. New York, NY, USA: ACM, 2004, pp. 59–67. [Online]. Available: http://doi.acm.org/10.1145/1012453.1012464

[38] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson, "The open provenance model: An overview," in *International Provenance and Annotation Workshop*. Springer, 2008, pp. 323–326.

[39] Y. Liu, J. Futrelle, J. Myers, A. Rodriguez, and R. Kooper, "A provenance-aware virtual sensor system using the open provenance model," in *2010 International Symposium on Collaborative Technologies and Systems*, May 2010, pp. 330–339.

[40] Y. L. Simmhan, B. Plale, and D. Gannon, "Karma2: Provenance management for data-driven workflows," *Web Services Research for Emerging Applications: Discoveries and Trends: Discoveries and Trends*, p. 317, 2010.

[41] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, *Workflow evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 438–455. [Online]. Available: http://dx.doi.org/10.1007/BFb0019939

[42] Y. Gil and S. Miles, *PROV Model Primer*, W3C Working Group, Apr. 2013, 30. [Online]. Available: https://www.w3.org/TR/prov-primer

[43] ——, "Prov model primer," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/

[44] P. Groth and L. Moreau, "Prov-overview," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

[45] L. Moreau and P. Missier, "Prov-dm: The prov data model," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-dm-20130430/

[46] T. D. Nies, "Constraints of the prov data model," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-constraints-20130430/

[47] T. D. Nies and S. Coppens, "Prov-dictionary: Modeling provenance for dictionary data structures," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-dictionary-20130430/

[48] H. Hua, C. Tilmes, and S. Zednik, "Prov-xml: The prov xml schema," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-xml-20130430/

[49] G. Klyne and P. Groth, "Prov-aq: Provenance access and query," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-aq-20130430/

[50] L. Moreau and P. Missier, "Prov-n: The provenance notation," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-n-20130430/

[51] "Ssemantic of the prov data model," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-sem-20130430/

[52] S. Miles, C. M. Trim, and M. Panzer, "Dublin core to prov mapping," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-dc-20130430/

[53] "Linking across provenance bundles," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-links-20130430/

[54] I. Suriarachchi, Q. G. Zhou, and B. Plale, "Komadu: A capture and visualization system for scientific data provenance," *Journal of Open Research Software*, vol. 3, no. 1, mar 2015. [Online]. Available: http://dx.doi.org/10.5334/jors.bq

[55] X. Ma, J. G. Zheng, J. C. Goldstein, S. Zednik, L. Fu, B. Duggan, S. M. Aulenbach, P. West, C. Tilmes, and P. Fox, "Ontology engineering in provenance enablement for the national climate assessment," *Environmental Modelling & Software*, vol. 61, pp. 191 – 205, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364815214002254

[56] C. Tilmes, P. Fox, X. Ma, D. L. McGuinness, A. P. Privette, A. Smith, A. Waple, S. Zednik, and J. G. Zheng, *Provenance Representation in the Global Change Information System (GCIS)*, ser. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, June 2012, vol. 7525, ch. Provenance and Annotation of Data and Processes, pp. 246–248. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34222-6_28

[57] X. Ma, P. Fox, C. Tilmes, K. Jacobs, and A. Waple, "Capturing provenance of global change information," *Nature Clim. Change*, vol. 4, no. 6, pp. 409–413, Jun 2014, commentary. [Online]. Available: http://dx.doi.org/10.1038/nclimate2141

[58] A. Capiluppi, P. Lago, and M. Morisio, "Evidences in the evolution of os projects through changelog analyses," in *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering*, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, Eds., May 2003, citation: Capiluppi, A., Lago, P., Morisio, M. (2003). ?Evidences in the evolution of OS projects through Changelog Analyses.? in Feller, P., Fitzgerald, B., Hissam, B. Lakhani, K. (eds.) Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering ICSE?03 International Conference on Software Engineering Portland, Oregon May 3-11, 2003. pp.19-24.. [Online]. Available: http://roar.uel.ac.uk/1037/

[59] D. German, "Automating the measurement of open source projects," in *In Proceedings of the 3rd Workshop on Open Source Software Engineering*, 2003, pp. 63–67.

[60] K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller, "Open-source change logs," *Empirical Softw. Engg.*, vol. 9, no. 3, pp. 197–210, Sep. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:EMSE.0000027779.70556.d0

[61] K. Herzig and A. Zeller, "Mining cause-effect-chains from version histories," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, Nov 2011, pp. 60–69.

[62] "Rdfa core 1.1 - third edition: Syntax and processing rules for embedding rdf through attributes," March 2015, accessed: December 24, 2016. [Online]. Available: http://www.w3.org/TR/2015/REC-rdfa-core-20150317/

[63] C. Bizer, K. Eckert, R. Meusel, H. Mühleisen, M. Schuhmacher, and J. Völker, *Deployment of RDFa, Microdata, and Microformats on the Web – A Quantitative Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 17–32. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-41338-4_2

[64] E. Ainy, P. Bourhis, S. B. Davidson, D. Deutch, and T. Milo, "Approximated summarization of data provenance," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '15. New York, NY, USA: ACM, 2015, pp. 483–492. [Online]. Available: http://doi.acm.org/10.1145/2806416.2806429

[65] B. Cao, Y. Li, and J. Yin, "Measuring similarity between gaphs based on the levenshtein distance," *Applied Mathematics & Information Sciences*, vol. 7, no. 1L, pp. 169–175, 2013.

[66] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and Applications*, vol. 13, no. 1, pp. 113–129, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10044-008-0141-y

[67] A. Hliaoutakis, G. Varelas, E. Voutsakis, E. G. M. Petrakis, and E. Milios, "Information retrieval by semantic similarity," in *Intern. Journal on Semantic Web and Information Systems (IJSWIS), 3(3):5573, July/Sept. 2006. Special Issue of Multimedia Semantics*, 2006.

[68] Y. Ma, M. Shi, and J. Wei, "Cost and accuracy aware scientific workflow retrieval based on distance measure," *Information Sciences*, vol. 314, no. C, pp. 1–13, Sep. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2015.03.055

[69] W. C. Tan, "Research problems in data provenance." *IEEE Data Eng. Bull.*, vol. 27, no. 4, pp. 45–52, 2004.

[70] 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.

[71] M. Klein and D. Fensel, "Ontology versioning on the semantic web," in *Stanford University*, 2001, pp. 75–91.

[72] M. S. Mayernik, T. DiLauro, R. Duerr, E. Metsger, A. E. Thessen, and G. S. Choudhury, "Data conservancy provenance, context, and lineage services: Key components for data preservation and curation," *Data Science Journal*, vol. 12, pp. 158–171, 2013.

[73] (2012, Jun.) Dcmi metadata terms. DCMI Usage Board. Accessed: February 8, 2017. [Online]. Available: http://dublincore.org/documents/2012/06/14/dcmi-terms/

[74] T. Stevens, "Nasa gcmd kkeyword version 8.4 released," Aug. 2016, accessed: February 10, 2017. [Online]. Available: https://wiki.earthdata.nasa.gov/display/CMR/NASA+GCMD+Keywords+Version+8.4+Released

[75] B. Polyak, E. Prasolov, I. Tolstikhin, L. Yakovlev, A. Ioffe, O. Kikvadze, O. Vereina, and M. Vetrina, "Noble gas isotope abundances in terrestrial fluids," 2015. [Online]. Available: https://info.deepcarbon.net/vivo/display/n6225

[76] S. Morrison, R. Downs, J. Golden, A. Pires, P. Fox, X. Ma, S. Zednik, A. Eleish, A. Prabhu, D. Hummer, C. Liu, M. Meyer, J. Ralph, G. Hystad, and R. Hazen, "Exploiting mineral data: applications to the diversity, distribution, and social networks of copper mineral," in *AGU Fall Meeting*, 2016.

[77] J. F. Roddick, "A model for schema versioning in temporal database systems," *Australian Computer Science Communications*, vol. 18, pp. 446–452, 1996.

[78] P. Klahold, G. Schlageter, and W. Wilkes, "A general model for version management in databases," in *Proceedings of the 12th International Conference on Very Large Data Bases*, ser. VLDB '86. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, pp. 319–327. [Online]. Available: http://dl.acm.org/citation.cfm?id=645913.671314

[79] S. Pröll and A. Rauber, "Citable by design - A model for making data in dynamic environments citable," in *DATA 2013 - Proceedings of the 2nd International Conference on Data Technologies and Applications, Reykjavík, Iceland, 29 - 31 July, 2013*, 2013, pp. 206–210. [Online]. Available: http://dx.doi.org/10.5220/0004589102060210

[80] M. Helfert, C. Francalanci, and J. Filipe, Eds., *DATA 2013 - Proceedings of the 2nd International Conference on Data Technologies and Applications, Reykjavík, Iceland, 29 - 31 July, 2013*. SciTePress, 2013.

[81] S. Proell and A. Rauber, "Scalable data citation in dynamic large databases: Model and reference implementation," in *IEEE International Conference on Big Data 2013 (IEEE BigData 2013)*, 10 2013.

[82] B. Lafuente, R. T. Downs, H. Yang, and N. Stone, "1. the power of databases: The RRUFF project," in *Highlights in Mineralogical Crystallography*, T. Armbruster and R. M. Danisi, Eds. Walter de Gruyter GmbH, 2015, pp. 1–30. [Online]. Available: http://dx.doi.org/10.1515/9783110417104-003

[83] C. Ochs, Y. Perl, J. Geller, M. Haendel, M. Brush, S. Arabandi, and S. Tu, "Summarizing and visualizing structural changes during the evolution of biomedical ontologies using a diff abstraction network," *J. of Biomedical Informatics*, vol. 56, no. C, pp. 127–144, Aug. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jbi.2015.05.018

[84] M. Hartung, A. Gro, and E. Rahm, "Contodiff: generation of complex evolution mappings for life science ontologies," *Journal of Biomedical Informatics*, vol. 46, no. 1, pp. 15 – 32, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532046412000627

[85] "Overview," accessed: February 14, 2017. [Online]. Available: https://nsidc.org/ssiii/

[86] Ssiii ontologies. National Snow & Ice Data Center. Accessed: February 14, 2017. [Online]. Available: https://nsidc.org/ssiii/ontology.html

[87] Z. B. Miled, S. Sikkupparbathyam, O. Bukhres, K. Nagendra, E. Lynch, M. Areal, L. Olsen, C. Gokey, D. Kendig, T. Northcutt, R. Cordova, G. Major, and N. Savage, "Global change master directory: Object-oriented active asynchronous transaction management in a federated environment using data agents," in *Proceedings of the 2001 ACM Symposium on Applied Computing*, ser. SAC '01. New York, NY, USA: ACM, 2001, pp. 207–214. [Online]. Available: http://doi.acm.org/10.1145/372202.372324

[88] "Keyword faq," Earthdata, 2016, accessed: December 12, 2016. [Online]. Available: https://wiki.earthdata.nasa.gov/display/CMR/Keyword+FAQ