DATASET VERSIONING THROUGH LINKED DATA MODELS

By

Benno Lee

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

_____
Peter Fox, Thesis Advisor

_____
Jim Hendler, Member

_____
Deborah MacGuiness, Member

_____
Beth Plale, Member

Rensselaer Polytechnic Institute
Troy, New York

May 2018
(For Graduation July 2018)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Listings

ACKNOWLEDGMENT

I would like to thank my committee for their input into my research journey. I would like to thank Peter Fox for his guidance and advice. I would like to thank Kathy Fontaine for her guidance and time in editing this document. I would like to thank my lab mates for insightful and lengthy discussions.

ABSTRACT

Data sets invariably require versioning systems to manage changes due to an imperfect collection environment. Data versioning systems are employed to manage changes to data, logging new data sets and communicating that change to data consumers. Versioning discussion remains imprecise, lacking standardization or formal specifications. Many works tend to define versions around examples and local characteristics but lack a broader foundation. This imprecision results in a reliance on change brackets and dot-decimal identifiers without quantitative measures to justify their application. No difference exists between the versioning practices of a group which updates their data regularly and a group which adds many new files but rarely replaces them. This work attempts to improve discussion by capturing version relationships into a linked data model, taking inspiration from provenance models that incorporate versioning concepts such as PROV and Provenance, Authorship, Versioning (PAV) ontologies. The model captures addition, invalidation, and modification relationships between versions to provide change log-like characterization of the differences.

The data set landscape largely lacks the practice of including detailed change documentation like the logs commonly accompanying software projects. The result likely originates from the size of data sets requiring time-intensive work to manually construct. A process is presented to automate change log creation for data sets, improving adoption as well as encoding the change logs with linked data. The linked data equipped change logs makes the documentation consumable by machines, ensuring not only efficient creation, but also utilization. A drawback, as found from larger data set change, the encoding causes significant bloating as compared to plain text change logs. The versioning graphs encoded into the document, describing the **additions**, **invalidations**, and **modifications** (AIM) made by the new version, allow new avenues of exploration into data that can be standardized across data sets.

The AIM changes allow versions within the same data set to be compared

using counts of the changes. The comparisons provide a quantifiable basis for communicating change as compared to the qualitative version identifiers currently used by the Global Change Master Directory (GCMD) Keywords. The analysis revealed conflicts between the observed change in the data set at Version 8.5's release depending upon the perspective of the data consumer or producers. Data from the Marine Biodiversity Virtual Laboratory was also explored to give new context to the linked data version comparison process. The changes computed provided evidence towards measurably better performance in marine biology classification using the versioning model.

Version change logs link an amount of change to a particular version, but versions do not have to be published at regular intervals. That observation points towards the idea that just looking at versions may obscure trends in change over time by breaking up changes by version. An analysis was performed to observe the change rates of each version distributed over time for the GCMD Keywords and discovered that the versions could be clustered according to different change behaviors. The Earth Observing Laboratories were also studied for trends over time to study a collection of similar data sets. The AIM changes revealed unexpected behaviors in the data sets with respect to the way versions were updated, showing a strong relation between adding and removing files. The distribution of changes over time were also compared to the general distribution of versions over time and revealed a significant difference in behavior.

Change analysis for data sets need a great amount of work as big data sets become more common. Terms and practices need to be standardize and formalized which begins with producing discoverable and consumable change documentation. The procedures explored showed promise using linked data models, but suffered from size bloating necessary to make the documents machine consumable. Once computable, producers can begin providing better quantitative measure for change in data, but analysis has shown that the perceived change may differ depending on the consumer of the data set. The experience highlights an obscured dynamic in change information between data producers and data consumer in which producers often dictate the means of evaluating version change. Diverging from version-primary

practices and including more detailed change accounting becomes a priority after discovering that versions can hide trends in the actual change rate. The difference between data set change rate and behavior suggests that future research is necessary to determine if the differences indicate versioning practices also need to be different.

CHAPTER 1
INTRODUCTION

1.1 Why Versioning is Important

> "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." [1]

Anyone who has used an iPhone or owned a video game console understands the basics of versioning. Companies brand sequential devices to indicate improvements in performance or capabilities. Basic numerical sequencing has given rise to a plethora of versioning systems used widely across a landscape of software and data. Versioning systems help scientific workflows avoid losing work by managing transitions and changes while in operation [2]. Versioning systems provide necessary documentation which informs the transition to new methods and procedures [3]. Versioning systems provide accountability for the value of a project's data set when considering an agency's continued funding [4]. The natural evolution of versioning systems, however, have given rise to formal architecture operating on top of very informal concepts. In this dissertation, we identify gaps in versioning practices which result from tradition and develop a data model to more completely capture the interactions involved in versioning.

We know that our instruments are imperfect, thus making our data also imperfect. The data, however, can be collected under quantifiable amounts of error for which makes the data still usable. When imperfections within the data exceed expectations, versioning systems allow exceptional errors to be managed. The data can be removed, corrected, or specially treated to bring the data error back within expectations.

At the very core, versioning systems are a means of communication. Data producers communicate to consumers how much the producer has changed the data.

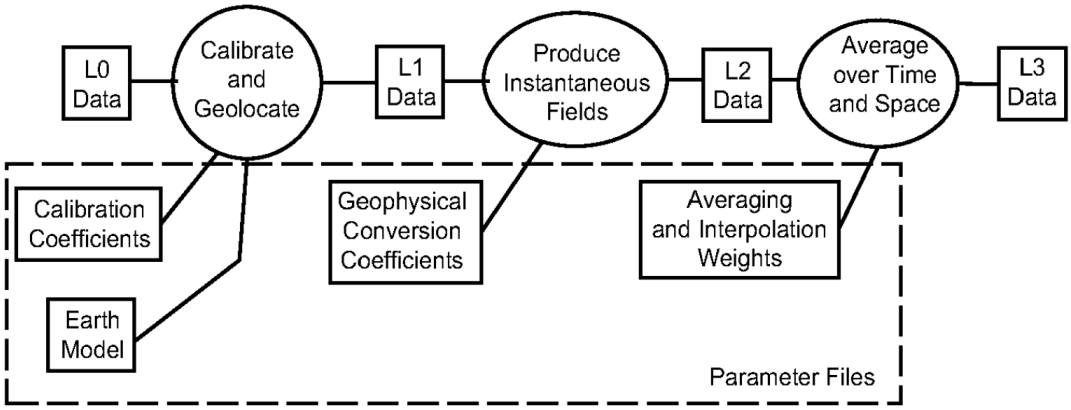


Figure 1.1: National Aeronatuics Space Administration Earth Science organizes its data into three levels depending on the amount of aggregation and the distance the data is removed from the original sensor measurements. Figure 1 from [1]

The producer can also communicate through time using logs and other documentation. In order to clearly communicate changes, ideas must be clearly formalized and defined.

1.2 Definitions of Version

Using the term 'versions' in the vernacular has become so pervasive that few documents formally define it. Barkstrom describes versions as **homogeneous groupings** used to control, "production volatility induced by changes in algorithms and coefficients as result of validation and reprocessing," [1]. The **groupings** he mentions is a method of separating data objects such that they have similar scientific or technical properties. In order to determine when these properties have changed, he leverages the NASA Earth Science workflow model shown in Figure 1.1. The model describes the formal stages of processing to turn a raw remote sensing signal from satellite instruments into global aggregate summaries [1]. Understanding this model reveals that changes to either the algorithms or parameter files will force a change in the resulting data, creating a new version of the output data. Essentially, versions are a means to communicate how much data has diverged as a result of changes to an object's provenance.

Another definition comes from Tagger in which versions are a, "semantically

meaningful snapshot of a design object," [5]. He, unfortunately, does not further clarify what he means by semantically meaningful. The design object unifies the versions as their primary subject, capturing the object's state over the course of its design.

The *derivation*, PROV Ontology's analog for a version and covered more in Section 1.4.2, is defined 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," [6]. In this view, a **version** exists in comparison to another object.

The Functional Requirements for Bibliographical Records (FRBR) avoids the terms **edition** and **version** since "those terms are neither clearly defined nor uniformly applied" [7]. Instead, they use the terms: **work**, **expression**, and **manifestation**. A **work** refers to the abstract concept of a creative or artistic idea. **Expressions** are then different forms of that particular **work**, embodying the most similar term to the previous definitions of versions. A **manifestation** is the physical embodiment of an **expression**. These three terms and their hierarchy establish a repeating theme throughout other versioning works.

Combining these myriad of definitions together, a version is "an **expression** of a **work** which exists in comparison to another object and communicates the extent to which it diverges from that object as a result of provenance changes". Although each definition disagrees on the form a version object takes on, all but PROV derivation agree that a version belongs to a larger collection of objects implementing a more abstract, ideal representation. Provenance provides the information necessary to explain semantically meaningful for the Tagger definition as *prov:Derivation* captures when a data object diverges into a new object.

1.3 Version Models

Version models provide a visual theoretical aid in understanding where a data object lies in relation to the rest of a work. The Atmospheric Radiation Measurements (ARM) group used a model dividing the data into mathematical sets which versioning operations acted upon[8]. Adding files already in the set created a new

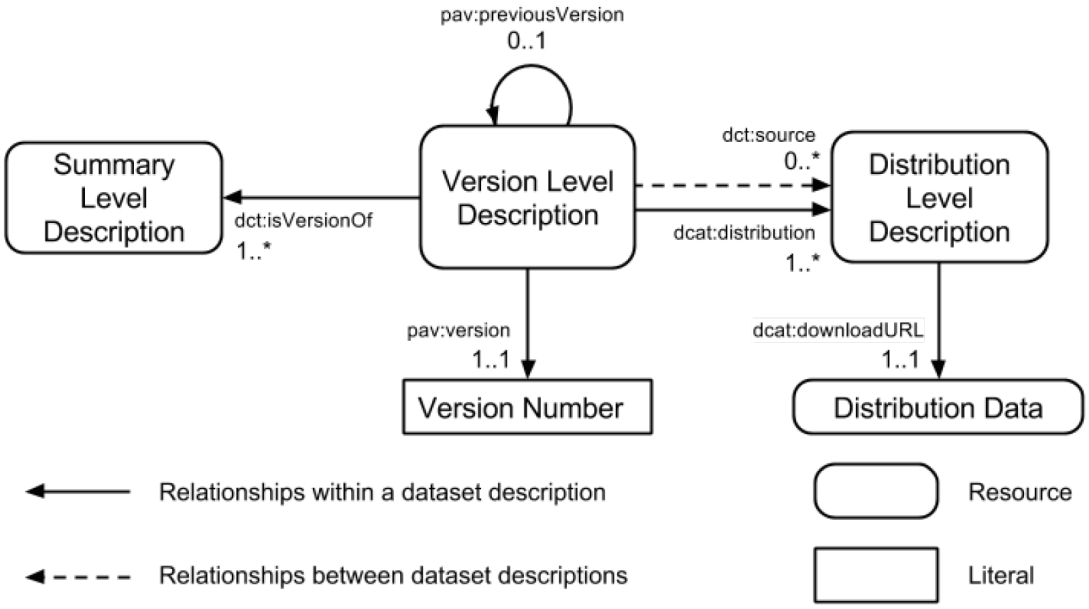


Figure 1.2: Data model from the Health Care and Life Sciences Interest Group separating data into three levels: works, versions, and instances. From Dummontier, et al. [9]

set which inherited all non-intersecting files and included all the new ones. The model provided a means to organize and automate the versioning of ARM's daily expanding data sets.

The Health Care and Life Sciences (HCLS) Interest Group of the World Wide Web Consortium (W3C) recently released a model which may provide a solution when used in conjunction with other identifiers [9]. Their model, shown in Figure 1.2, separates the concept of a data set into three groupings. The highest level summarizes the data as an abstract work, perhaps better described as a topic or title. The data topic can have multiple versions over time. The version can then be instantiated into various distributions with different physical formats. The model—relating summary, version, and distribution—also strongly resembles the formation of FRBR's work, expression, and manifestation model.

From his definition of versions, Barkstrom also outlines an hierarchical version model as seen in Figure 1.3. The model features additional intermediary levels than the HCLS's model, following NASA's data curation practices [10]. Each edge in the tree signifies a difference with other objects at the same depth, but the

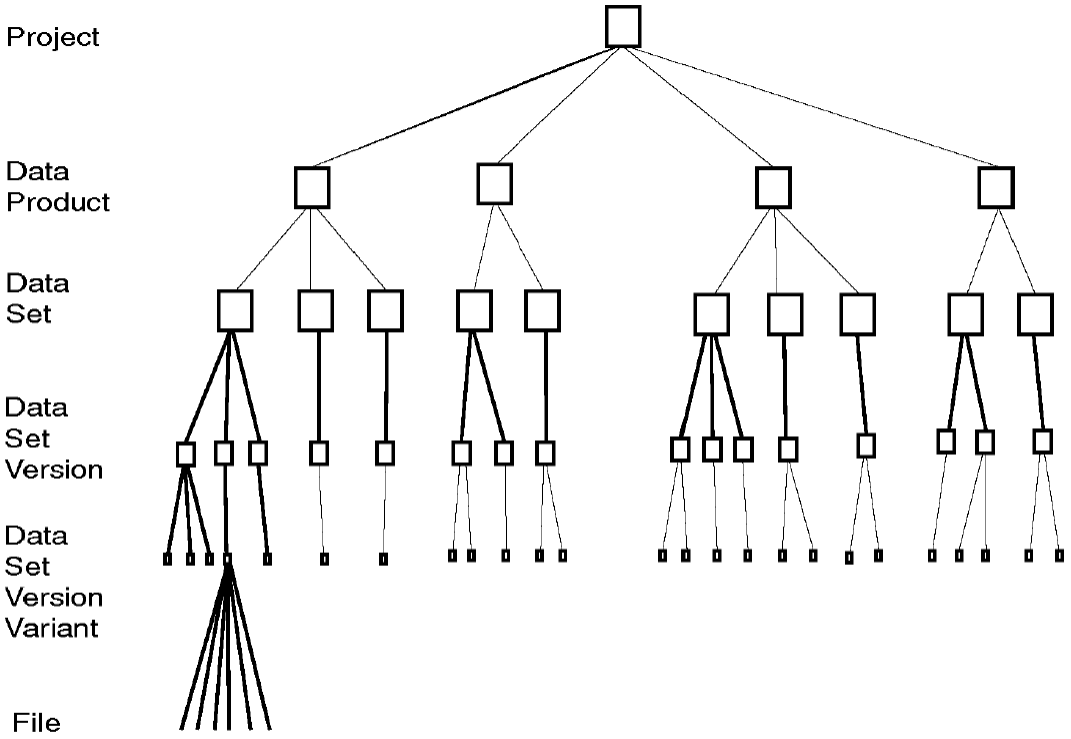


Fig. 2. Hierarchical groupings of files from data products to individual files

Figure 1.3: Visual representation of grouping hierarchy. From [1]

model does not provide a mechanism to explain the difference. The difference in the number of tiers employed in the HCLS and Barkstrom models also indicates that different applications will have varying expectations of granularity to their versioning models. A general solution will likely need to be tiered and recursive in structure to accommodate different levels of specificity.

1.4 Provenance Representation

Provenance ontologies form a major section of linked data approaches to data versioning. The coverage stems from the close relation between provenance and differentiating versions. The Proof Markup Language, one of the first semantic models to capture provenance information, expressed lineage relationships using inference reasoning through traceable graphs [11]. The technique provides a powerful

way to express and imply sequences of relationships between different versions and characterize the manner of their relation.

1.4.1 Open Provenance Model

A number of linked data models include versioning concepts such as the Open Provenance Model (OPM) [12]. Driven by the uncertain needs and sometimes conflicting conventions of different scientific domains, the model sought to find a method to standardize 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, automating and unifying their provenance capture even as they grow [13]. To aid OPM's adoption, the framework Karma2 integrates provenance capture into scientific workflows and provides a more abstract view of their data collection activities [14]. The property *opm:WasDerivedFrom* constitutes a core concept in the model and marks the reliance of one object's existence on another object. For a large part, this encompasses the engagement which provenance models view versions, without further need to explore the derivation's content.

1.4.2 PROV-O

PROV, a World Wide Web Consortium (W3C) Recommendation, delineates a method to express data provenance in a more compact form as seen in Figure 1.4 [15] [16]. The recommendation uses a conceptual model relating activities, agents, and entities to describe data production lineage [17] [18] [19]. Intended as a high level abstraction, it takes an activity-oriented approach to provenance modeling. Every data entity results from the actions of some activity [20]. The conceptual model's expression occurs through the PROV Ontology (PROV-O), which can be conveyed through various resource description languages [21] [22]. The ontology is further formalized into a functional notation for easier human consumption [23] [24]. One particular strength that has contributed to the adoption of PROV is its ability to link into other ontologies, making it easier for existing semantically enriched data sets to adopt PROV [25] [26].

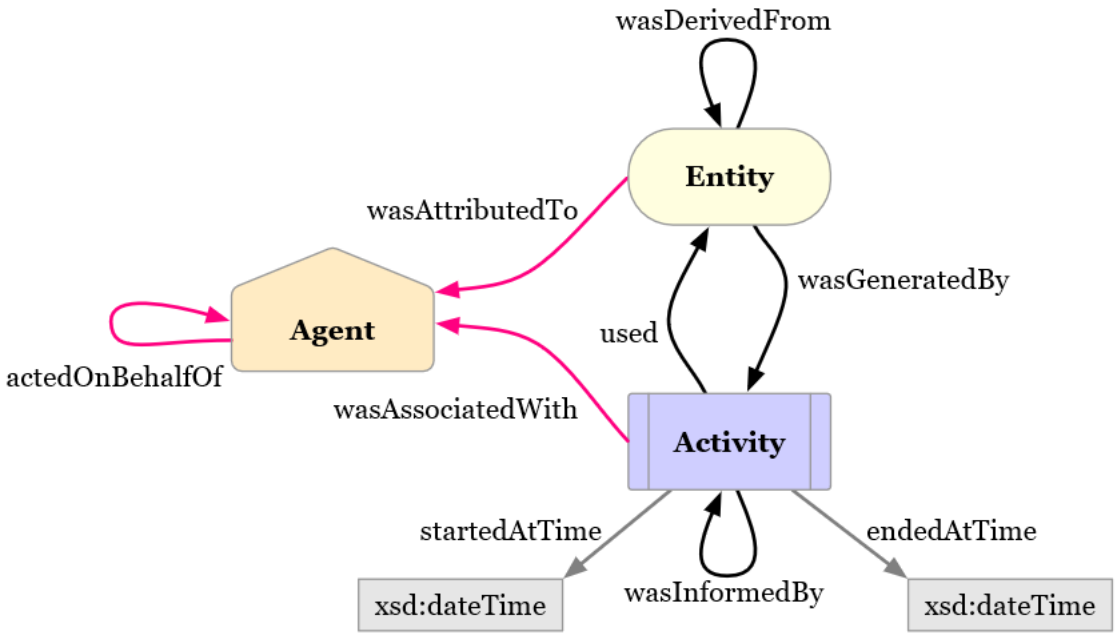


Figure 1.4: Diagram of the PROV Ontology. Figure 1 from [6]

PROV has provided a major contribution in maintaining the quality and reproducibility of data sets and reporting in the National Climate Assessment (NCA) [27]. The contribution 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 assessment report [28]. The usage means that not only does the data have a traceable lineage to verify quality, but the content of documents can have the same verifiability [29]. Komadu, a framework developed to alleviate workflow integration, utilizes PROV to improve upon its predecessor, Karma, by no longer utilizing global context identifiers that were not necessarily shared throughout the workflow. [30].

The PROV Ontology provides three different concepts that begin to encapsulate the provenance relationship between data versions. It defines a *prov:Generation* as "the completion of production of a new entity by an activity," [6]. This means that the generation, which corresponds adding an object to a version, must result from a *prov:Activity*. *Prov:Invalidation*, defined as the, "start of the destruction, cessation, or expiry of an existing entity by an activity," makes a similar connection

between activities and entities [6]. A third concept, *prov:Derivation*, relates 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 preexisting entity. " [6]. PROV also has a property called *prov:isDerivedFrom* which conveys the same definition as a *prov:Derivation*. Using the property and concept together forms a qualified property which can be instantiated and further annotated.

1.4.3 Provenance, Authorship, and Versioning Ontology

The Provenance, Authorship, and Versioning (PAV) Ontology is, "a lightweight vocabulary, for capturing "just enough" descriptions essential for web resources representing digitized knowledge" [31]. It 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 [31]. 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" [32]. PAV supports the idea 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. While it documents related versions well, PAV does not dive deeper in explaining the circumstances behind version differences.

1.4.4 Schema.org

The Schema.org ontology is not a provenance ontology but provides a means to supply searchable web pages with standardized micro-data. The ontology has a collection of concepts which could be applied to versioning. The *schema:UpdateAction* is defined as, "the act of managing by changing/editing the state of the object," which encompasses the same responsibilities expected of versioning systems [33]. The terms *schema:AddAction*, *schema:DeleteAction*, and *schema:ReplaceAction* subclass the *shcema:UpdateAction*. These classes model actions which further cement parallels between versioning and *schema:UpdateAction*.

Schema.org defines a *schema:ReplaceAction* as, "the act of editing a recipient by replacing an old object with a new object" [34]. The concept has two properties,

schema:replacee and *schema:replacer* which indicates that a new object replaces an old one. Schema.org models the interaction by placing the replacement action at the relation's center. In comparison, the *schema:AddAction* is defined as, "the act of editing by adding an object to a collection" [35]. The action only involves the object and the new state of the collection, not involving any of the collection's prior lineage. Schema.org defines the *schema:DeleteAction* as, "the act of editing a recipient by removing one of its objects," [36]. The concept aligns well with other versioning systems, although deletion may be a strong assertion.

1.5 Documenting Versions

Change logs, artifacts resulting from the versioning process, play a major role filling in gaps between versions. The logs document changes and explain, in human language, motivations behind the modifications [37]. Since identifiers denote that a change has occurred, the logs provide details on how the changes modify an object's attributes. They demonstrate a need and utility in understanding the deeper content of change beyond knowing that an object did transform. While some data sets will contain a change log, software projects have normalized their use in version release documentation. As a result, these projects provide a basis for understanding the value of change logs to data sets with multiple versions. The change log's common drawback is the limitation to only human readable text. Wider adoption among data sets may be possible by making these texts machine computable.

Open source projects use change logs more consistently than data projects, which usually sport only use documentation [38]. Logs play an important communication role in these projects since developers can contribute without having been part of the original development team. The change logs allow developers to link bugs and errors with their corrections in new versions of the code [39]. The links gives insight into motivations behind particular design decisions. Logs linked with version releases also provide feedback to the user community that corrections have been addressed, in addition to ensuring that improvements drive modifications to the code base. An identifier cannot communicate these qualities while remaining succinct. Some research has been done to determine the health of a development

project based on the number and length of change logs released over time [38]. Little work has been done to make change logs machine-computable, as many of these documents remain in human-readable text only. Research done involving change log content must manually link entries with computable meta-data such as the introduction of new features with the emergence of new bugs [40]. 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 data sets is then left largely up to the human user's ability to understand and process the modifications mentioned within the change log.

1.6 The Versioning Use Case

The research questions addressed by work in the following chapters revolve around different aspects of the flow in use case shown in Table 1.1. The use case begins when a data producer wishes to distribute the next version of a data set. While versions for observations of phenomena is often sequential, alternate versions can deal with older data or occur concurrently with the object the alternate changes so the adjective new or newer is avoided when possible. The use case features the data producer and data consumer as actors using the versioning system as an intermediary. The actors do not follow separate use cases because each actor relies on the actions of the other actor to contextualize the actions in the use case.

The basic flow of the use case is seen in Figure 1.5 where the producer logs a version into the versioning system, and the consumer retrieves that data version. At a later point in time, the producer can add any number of corrections to the versioning system. The repeated corrections reflect a practice of passively logging additional changes. At some final time after 0 or more corrections, the Consumer returns to check the versioning system for any changes to the data set the Consumer retrieved. If there are changes to parts pertinent to the Consumer, the actor retrieves the corrected data. In the interaction, the Producer only passively provides additional versions of the data while the responsibility of remaining up-to-date lies

Table 1.1: Versioning Use Case Table

Use Case Name: New Version Publication & Retrieval
Goal: Record a new version of a data set and provide it to a data consumer.
Summary: The Producer creates a new version of their data and must record it to the Versioning System while providing the Consumer with the data
Actors: Producer, Consumer
Preconditions: Producer has supplied some data to the Versioning System. Consumer has retrieved the data from the Versioning System.
Triggers: Producer provides a different version of the data to the Versioning System.
Basic Flow: 1. Producer places the next version into the Versioning System. 2. Producer may repeat the previous action 0 or more times. 3. Consumer checks the data to see if there are changes. 4. If there are pertinent changes: (a) Consumer retrieves the updated data.
Alternate Flow: 1. Producer places the next version into the Versioning System. 2. Producer notifies Consumer that an alternate version is available. 3. Consumer retrieves the updated data.
Post Conditions: Producer has made the alternate version available. Consumer possesses the pertinent newly available version.

with the Consumer. The ability of the versioning system to communicate to the Consumer that the data has changed determines whether the use case succeeds or fails for the data consumer. The relationship creates a Producer/Consumer dynamic which influences the performance of the versioning system.

The alternate flow in Figure 1.6 requires additional information from the Consumer which is a means of notification. The flow begins the same as the basic flow, but after a single correction, the Producer notifies the Consumer that the data has changed. The notification causes the Consumer to come and retrieve the updated

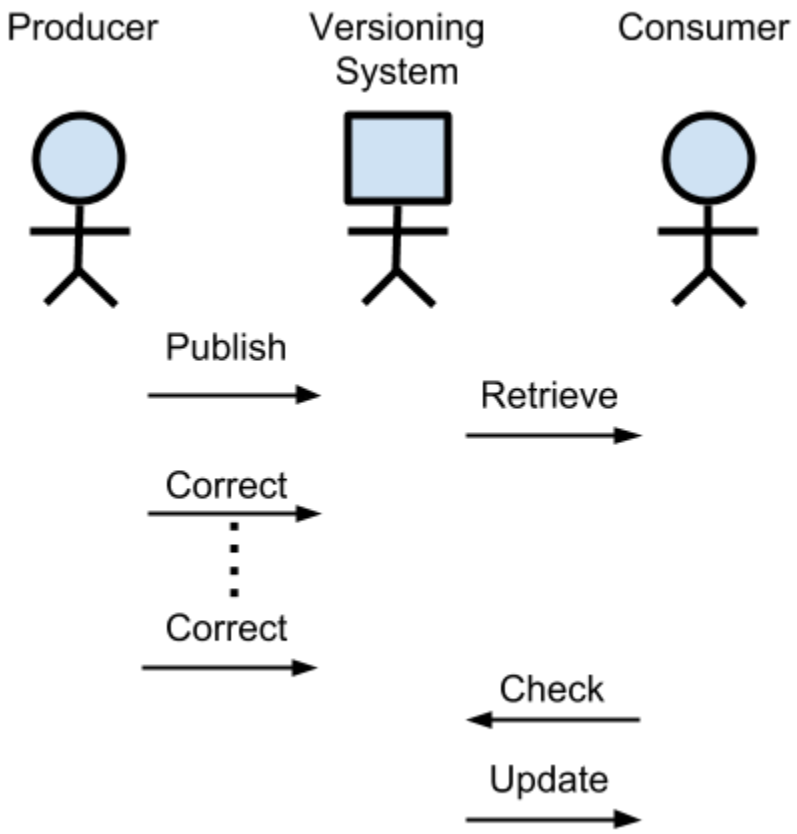


Figure 1.5: Basic Flow Use Case Diagram.

data set. The alternate flow poses a few problems in that the Producer must now manage notification information, but the Producer must also notify all consumers of the data which may cause scalability issues.

1.6.1 Research Question 1: What has changed?

The primary research question addressed in the following chapters pertains to step 3 in the basic flow of the use case. In order to execute this step, the Consumer must have a means of determining whether the data set is different from the one the Consumer currently possesses. In a large number of software projects, especially open-source projects, change logs document for the producer changes to the code base and communicate alterations to the Consumer. Very few data sets provide detailed change logs along with versions in the versioning system. Either general

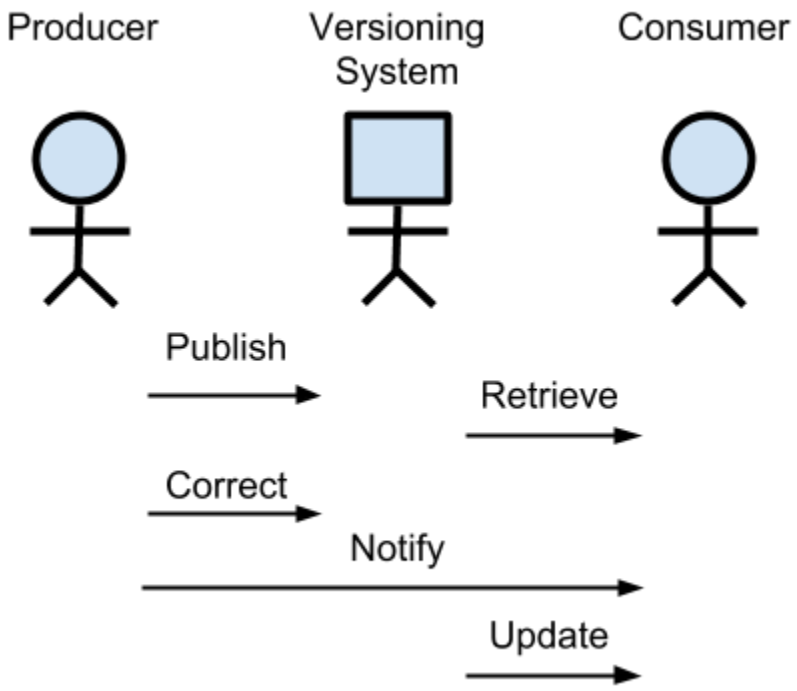


Figure 1.6: Alternate Flow Use Case Diagram.

descriptions of changes are made or new usage documentation is provided, requiring the data consumer to manually acclimate to changes.

The prevailing hypothesis in addressing Research Question 1 is that automating the generation of change logs will provide detailed change documentation for even large data sets. Different standards exist across various national agencies, but few fundamentals have been established on what needs to be included and how the information is to be captured or presented. A linked data versioning model would provide a standardized basis for discussing version change as well as allow tools to be developed to assist in step 3 for large data sets of which Consumers may only use a part.

1.6.2 Research Question 2: How much has changed?

A follow-up query to Research Question 1 is that once the changes have been determined, Consumers need to evaluate the impact of significance of the changes. At the moment, Producers regularly make dot-decimal identifiers pull double duty

in organizing versions sequentially and implying change magnitude. Version counts, as shown in Chapter 4, provide a very bland description of the changes in a version, lacking nuance and effected by multiple outside practices. Using the versioning model to answer Research Question 1, the changes can be broken down into **Add**, **Invalidate**, and **Modify** (AIM) classifications which provide a more encapsulating description of a version transition. The AIM change counts would tie change metrics back to data differences rather than to versions. The AIM changes also provide a means to compare the difference between the amount of change declared by the Producer and quantity of change as seen by various Consumers.

1.6.3 Research Question 3: How fast does the data change?

An important idea to understand is how often will the consumer need to execute the versioning use case. Research takes time and if the data is unstable, new versions of the data may be produced before results can be extracted from the data, invalidating the findings. With the ability to measure changes from Research Question 2, changes in each version can be tied back to time. Versions package and release changes in a bundle, disconnecting the changes from time but limiting the data volatility to an acceptable level. The hypothesis is that versions are masking the actual change rate, the giving a false sense to the expected change in the data over time.

1.7 Hypothesis Statement

The work in this dissertation seeks to prove three hypotheses. In the first hypothesis, provenance concepts for revisions need to be diversified in order to properly capture change information. Prior versioning models indicate that versioning graphs will need to utilize a tiered approach to capture the relation between objects and their more granular attributes. These tiers unify the practices already established by provenance models with the semantics defined by versioning models. The content and format of change logs give insight into the information desired by version users. The versioning graphs in this dissertation will explicitly deal with data sets since software versioning is a more developed field. An additional assumption in

constructing the versioning graph is that the objects used in the comparison have already been established as versions of each other. That is, objects in the versioning graph belong to the same **work** and share the same tier.

The second hypothesis is that versioning graphs can calculate a more accurate change distance by counting the number of changes. Current methods rely on vague data producer evaluation, communicated through changing version identifiers. Because version graphs will catalog individual changes, a count based on the different types of changes are expected to produce a more revealing change metric. The distance can be verified by comparison with the amount of change indicated by the version identifier. The hypothesis relies on an assumption that all changes of the same type within a work have the same or similar significance. The evaluation stems from the observation that change logs exist to capture the differences between sequential versions.

The third hypothesis is that versioning graphs will enable the automated creation of machine readable change logs. Assuming that natural language processing cannot understand and quantify the change in a log, quantifying the log's information remains difficult. The results will show that a comprehensive log can be generated which is both human and machine consumable. The approach will deal specifically with spreadsheet data sets since they provide a flatter, more consistent context to capture.

1.8 Contributions

In Chapter 4, I develop my first contribution, the idea of a versioning graph, through the process of addressing Use Case 1. Versioning graphs capture differences between objects, not the course used to create those objects, differentiating themselves from provenance graphs. The versioning graph enables my second contribution, a process to automate machine-readable change log creation covered in Chapter 3. The contribution eases consuming very lengthy logs, which data sets often produce, as well as enabling searchability and discoverability of changes affecting the version. My third contribution, discussed in Chapter 4.3 is using a versioning graph to provide a quantitative basis for determining change distance. The resulting

distance measure comes in three parts to both characterize the distance and provide a basis for version identifier assignment.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

The data versioning landscape produces a variety of different approaches and standards towards change capture. Science agencies and organizations are only beginning to formally codify and standardize methods to capture and publish lineage information [41]. In comparing their methods, many systems also share the implementation of common versioning operations, suggesting an avenue for fundamental versioning properties. While Software Versioning Managers (SVM) prefer to adopt the dot-decimal identifier, Digital Object Identifiers (DOI) and other web identifiers contribute methods to connecting more expressive change documents. Change logs are a feature which commonly appears alongside software projects and provide insight in differences between versions, but they are found very rarely among data sets. Measuring the space between versions also appears under-explored in previous approaches.

2.2 Version Systems

Versioning systems take many different forms from Clotho, an application conducting versioning at the block level, to Champagne, a framework to propagate change data across multiple information systems [42] [43]. Each approach has a unique set of challenges to overcome. Closer to the data collection, version systems must be flexible and responsive to adapt to changing environments, but as the socio-technical distance of a repository increases away from the collection site, more formal methods are required to unify repositories [44]. Different approaches are also necessary to account for the needs of different domains. Versioning an XML text-file will need to account for serial file input and output as well as structured markup [45]. Many applications have adopted a tree-like structure which is further propagated by software versioning managers (SVM) [46]. The advantage comes from using well established graph theory methods, and applying the methods to an object's

complex relationships in complex environments [47]. The growing population of web documents, however, presents a new smrgsbord of complicated data which will need scalable solutions [48].

2.2.1 Library Sciences

While many of the modern systems requiring versioning managers store digital products, libraries have been tackling similar issues for a much longer time. Libraries curate multiple editions of the same work, sometimes with significant revisions [3]. In many ways, versioned objects resemble multi-edition books or documents. Digital librarians have faced many challenges when searching for a persistent identifier due to evolving web technologies. Early citations referred to on-line documents using stagnant Uniform Resource Locators (URL), but this frequently lead to a condition known as link rot where moving the document would invalidate the URL [49]. Locators required a system to manage changes of old identifiers to new locations when people attempted to utilize references from print. The need eventually led to the development of Persistent URLs (PURL), which also suffered from link rot, and this eventually led to the distributed Digital Object Identifier (DOI) system used to track documents today [50]. The PURL used a centralized system that would translate dead links and redirect to a document's latest location. The system would still need to be manually updated, meaning links would rot if a document was lost or overlooked. DOIs rely on a network of managing agencies to collect and host submitted documents. In the specialized Handle system, the network has member agencies internally assign an unique name and concatenate it to the end of their host name. In Figure 2.1, DOIs represent the most suitable identifier used for citation in scholarly literature [50]. The DOI network provides a robust system to track documents, but when tracking data, it faces difficulty following the rate of change with more volatile data sets. Under current definitions, distribution organizations assign different DOIs to separate editions of a document. Documents often do not need new identifiers since they change very rarely as a result of the publication process. Data set production and distribution cycles move more quickly and react more sensitively to small content changes, including when data collection continues

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 2.1: Table of predominant identifiers used in science. From Duerr, et al. [50]

on after initial publication. Data set behavior becomes entirely too slow as data providers begin allowing users to dynamically generate data products from existing data according to their needs [51]. Some agencies have begun assigning versioned DOIs, but this has not become common practice. Other groups do not assign a new DOI, but reference the latest release of the document or object [52].

As digital methods have evolved, so have digital libraries. The documents that digital libraries store are no longer constrained by physical organization [53]. A book can physically be randomly stored for efficient retrieval, but the digital copy may reside in multiple locations depending on dynamic filters or search queries. The Mellon Fedora Project developed a standardized edition control structure to unify disparate digital library stores [54]. The regularizing edition tracking methods significantly improved the response time and relevancy of the library services.

2.2.2 Software Versioning

Software versions form the most visible displays of versioning often experienced by researchers. Version managers provide tools to archive and restore code through the development lifecycle. The Revision Control System (RCS), developed in originally in 1985, documents one of the earliest uses of the dot-decimal identifier [55].

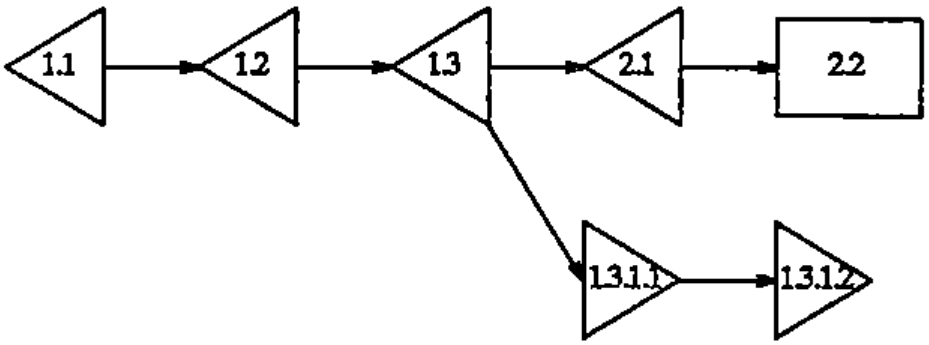


Figure 2.2: Commit history of an object in RCS with changes in the main line stored as back deltas and side branches stored as forward deltas. Figure 5 in [55]

This identifier uses a sequence of whole numbers concatenated by decimals. The system possessed many features of modern SVMs such as branches, a separate copy of the code for developing changes safely, which were identified by extending the dot-decimal identifier as seen in Figure 2.2. Not long after, the Concurrent Versions System (CVS) gained popularity with methods allowing multiple users to concurrently develop code to a central repository [56]. The most popular modern SVM is GIT which also allows concurrent development but enables distributed repositories [57]. Each developer contributing to a project is considered by the system to possess the master copy of that project. The users collaborate by requesting and pulling other developer's master copies into their project. In previous SVMs, only the differences between software files were stored, but GIT stores the entirety of each file version. Figure 2.3 demonstrates an example of how GIT employs storage space for multiple versions [57]. Only a pointer is stored in subsequent versions for unchanged files, saving space. Fischer, et al., demonstrate the importance of software version systems by integrating the manager with a bug tracking system to indicate the bugs a version release addresses [58].

2.2.3 Database Versioning

The need for dta versioning methods grew alongside the growing popularity and power of relational databases. Klahold, et al., introduced using abstract versioning environments in 1986 to separate the temporal features and organize the data

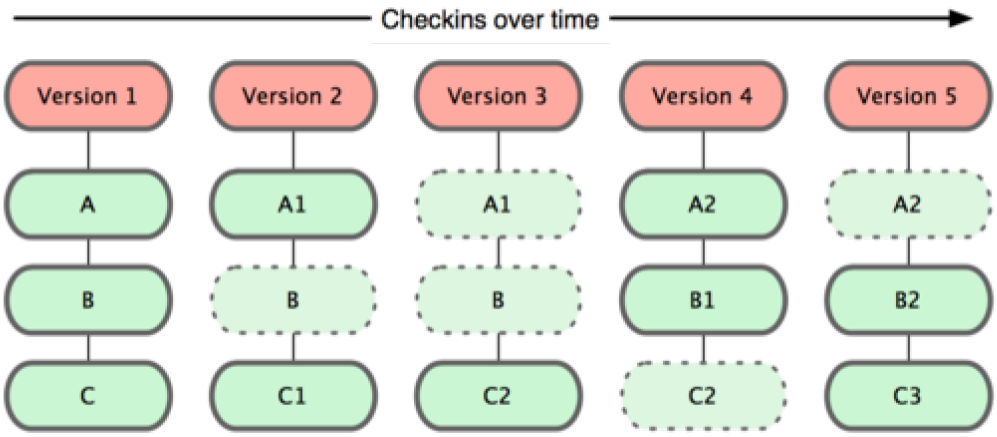


Figure 2.3: GIT stores changes in the repository as snapshots of individual files. Figure 1.5 from [57]

into related groupings [59]. Research in the versioning area focused primarily on the database schema. The results were temporal databases where schemas included time and dated transactions modifying the schema [60]. Temporal databases allowed old queries to be executed on updated schemas, improving the reproducibility of results. Capturing periodic snapshots or copies becomes unfeasible with increasingly large centralized database systems. Data collection continues to migrate towards massive data warehouses which store and serve a wide variety of data [61]. Proell and Rauber have investigated tracking data queries instead of the database as a more scalable solution to reproduce data [62]. The queries can then be used as publication citations to provide scalable, reproducible references to older data [63] [64].

2.2.4 Grid Versioning

The grid provides a sensitive environment for versioning where there are many users and data movement across the grid should be avoided. The CERN grid for the Compact Muon Solenoid experiment carefully developed processes which allow references by multiple users to the same file without copying that file across the grid [65]. Versions lock and release to permit parallel processing while still archiving additions and modifications to the data. Grid versioning applications also begins to highlight the difference in versioning usage patterns between users and producers [66]. Deeper exploration into the ATLAS system documentation did not reveal

specific use cases explaining the differences. The grid also provides users with the ability to begin dynamically defining data sets to their needs by aggregating results from across the network [51]. The process would create new data sets without prior existing change documentation and fueled a demand for responsive frameworks which could track the discordant data collection conditions assimilated by the system [67].

2.2.5 Ontology Versioning

Ontologies play a major role in defining domains, especially in the biological and medical fields where terms and definitions can change rapidly across highly variable organisms [68]. As a result, the ontologies require consistent methods to capture and model changes to evolving terms. Tools aid in the process by detecting differences between ontologies [69]. Klein and Fensel have found that when the changes are discovered, both forward and backward compatibility must be established for clear ontology versioning [70]. Not only must the path from an old term to a new one be clear, but a method for new terms to interact with old data must also exist. They additionally identified three levels at which ontologies can differ: the domain, the conceptualization, and the specification. Hauptmann et al., define a method to version ontologies natively within a triple store using linked data [71] [72]. The method heavily relies on the context of stored data.

2.2.6 Evaluation

Versioning systems cover a wide variety of different application environments, and each uses terminology to define versions in the context of their particular domain. Application based systems such as software and grid versioning focus primarily on identifying large, medium, and small differences between versions. The size approach suffers many drawbacks as a result of variety in versioning environments. Small changes logged in Clotho would barely register in massive systems operating on the grid. The requirements to differentiate changes is not universal across versioning systems. Other than software version managers, the systems do not incorporate methods to include change logs. They use the existence of an alternate version as sufficient explanation for what has changed. Bose, Frew, and Tagger all

recognize the need in versioning for a standardized representation, but each domain defines change according to the needs of their application [73] [5]. In isolation, the systems to not recognize the commonality in utilizing similar operations to conduct versioning activities.

2.3 Data Versioning Operations

Among all the systems surveyed in Section 2.2, every one employed some form of the operations add, delete, and modify. Literature surveys often expect versioning systems to interact with data uniformly because they are asked to perform the same functions [5]. Different data sets, however, may utilize each of the three core operations at different rates [74]. The differences help to characterize the data set in ways such as a growing set with many additions, a stable collection featuring occasional corrections, or a wildly volatile data set consisting of often deleted and replaced data files. Understanding these would give insight into the maturity and health of a data set.

While data addition and modification remain fairly uncontroversial, there is a mild division between practical and theoretical approaches to data deletion [42]. A removed object provides evidence of an erroneous activity's results or intermediary steps leading to a final product. As a result, version management should maintain and track invalidated data instead of deleting it. The software versioning manager GIT uses a method of compressing older data to conserve space without deleting the data [57]. Available storage space places pragmatic constraints on the number of projects which can adopt snapshotting practices. In applications which cannot recover erroneous data nor use it as documentation artifacts, like corrupted surveillance images. Some high energy physics experiments cannot re-collect observational data due to cost, and as a result, they cannot replace or re-process poor quality data [4]. While the distinction between 'deletion' and 'invalidations' remains largely semantic, the terms' use in this document reflects an understanding of the different constraints and requirements placed on versioning systems. As a result, invalidation is adopted as a broad, general term to also encompass data deletions.

A handful of other operations exist among version managers, but they do not

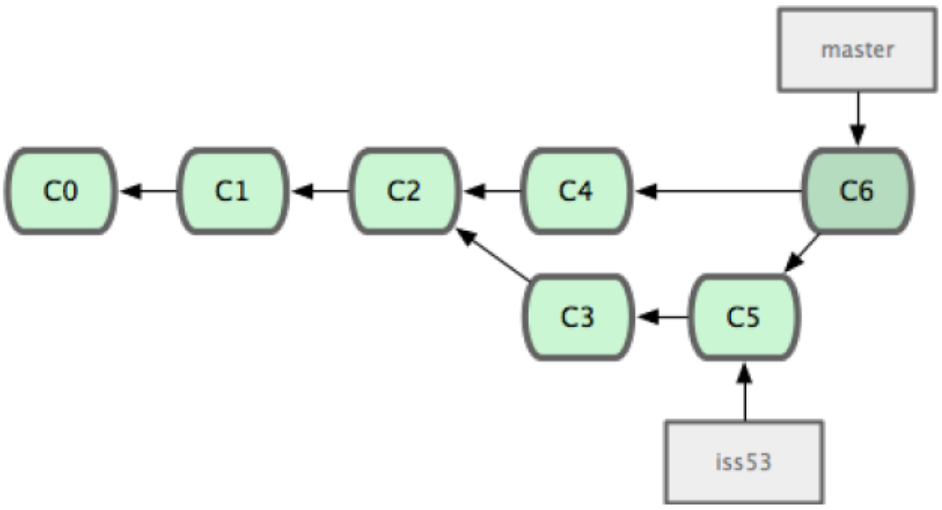


Figure 2.4: Example of a commit history with branching stored in GIT. Figure 3.17 from [57]

prove ubiquitous across most applications. Software versioning tools like RCS commonly feature branching and merging functions to create a versioning line separate from the stable master branch [55]. Branching mostly provides an organizational role in development by allowing developers to experiment without contaminating a stable software release. Figure 2.4 models a branching operation, showing versions C3 and C5 in branch iss53 before being merged back into the production line as C6. Branching allows for more orderly management of versions, but does not conduct versioning itself. Other activities provide functional operations such as locking and unlocking files from edits to prevent race conditions in branch mergers. Locks does not introduce any new relationships but allows the tool to operate more smoothly. Many version control tools, likewise, include functions to display the versioning tree, but this is also an ease-of-use function [47].

2.3.1 Types of Change

Another commonality across many versioning systems is differentiating between major, minor, and revision changes. Definitions for what constitutes each category differs across applications, but the desire to do so often stems from the tradition of 3-number dot-decimal identifiers. Barkstrom uses the ability to scientifically distinguish between two data sets as a criteria for major divisions among

groupings [1]. At lower levels, he notes that science teams can no longer discern scientific differences between data sets. They observe that, instead, changes to format and structure contribute significant alterations without changing any values withing the data. As a result, these technical changes form a second boundary to meaningfully separate minor version groupings. Finally, the explicit values may need occasional revisions to correct lexical errors such as spelling or formatting. Data producers will often use qualitative measures to determine the type of change occurring between versions. Versioning system users wish to achieve insight into the type of change that occurs between versions.

The exact category that a particular change falls into can be controversial. The decision to provide concentration units from parts per million to milligrams per milliliter poses a Technical change for a data producer. However, for a data consumer, the alteration may be viewed as a Scientific change as it invalidates the methods they had previously used. The conflict in view illustrates the data consumer-producer dynamic. In general, data producers control the versioning methods, but data consumers determine a change's impact through use. Producers tend to use versioning systems to ensure data quality of service through audits and recovery tools [4]. Meanwhile, a consumer will analyze the historical changes and determine the impact this may have on 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.

Version managers often disagree at the point many technical changes sufficiently modifies a data set that it comprises a scientific change. As determining changes in science requires expert understanding over a domain, different measures should be explored to address the distinction.

2.4 Identifiers

The most widely identifier scheme associated with versioning is the dot-decimal identifier [46]. Whenever, a new version is made, it receives an identifier with one of the numbers incremented as seen in Figure 2.2. Such a procedure fails to communicate the extent of a change because, regardless of the amount, the identifier will in-

crement only one number. Changes to the left-most number often signify a more important change. Many software applications use the 3-number Major.minor.revision format in labeling software releases. 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 not much value exists beyond that [47]. Most importantly, it groups together changes from the lower spectrum of minor or major change with those in the upper, more impactful, changes. Obtaining a clear characterization of a version change is difficult without a longer series of numbers. In addition, version numbers capture the overall change of a data set, but users may not interact with collections that way, only caring about parts of the data or certain kinds of change. There is also little standardization or formal requirements in naming methods. Ubuntu utilizes a dot-decimal version labeling scheme where the two number identifier corresponds to the year-month values of the release [75]. A common method used to address the distinction between versions is a human-readable change log, further discussed in Section 1.5.

The discourse on DOIs highlights the importance of understanding the limitations of particular identifier schemes. With respect to Figure 2.1, 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" [50]. A possibility to consider is that identifiers may require incorporation into a data model to discern between scientific differences. An identifier works well in revealing the characteristics of an individual object, but it should not be expected to explain its relationship with other objects. A data model provides better insight into the different roles objects play in a relationship. DOIs also provide a new means to identify versions using URIs which can be dereferenced to provide change information or the data depending on the context.

Using identifiers to convey extended versioning information becomes more difficult with the adoption of distributed version managers like GIT [56]. Each participant in the federated repository is the master of their personal copy of the code. Upon completion of their distribution's part, they may request that it be pulled

Figure 5.3: Benevolent dictator workflow

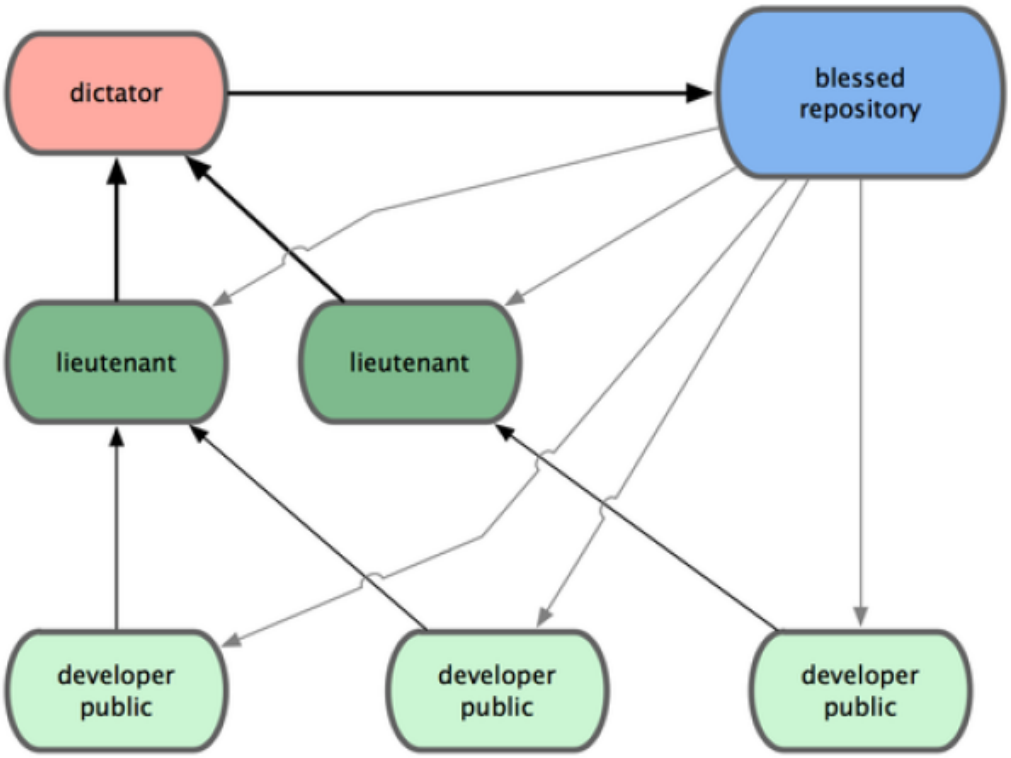


Figure 2.5: A distributed workflow to control for volatile versioning behavior. From [56].

into another participant's distribution. While each developer's individual repository can follow a linear identifier scheme, the identifiers would not work as the overall project bounces around different primary repositories with mismatching sequential identifiers. The dot-decimal identifier scheme could be made to work in such an environment by severely limiting the distributed manager's utilized features. Figure 2.5 illustrates a workflow which utilizes distributed repositories to manage very active public software projects. Each lieutenant developer manages a section of the overall code, and they dampen the number of requests made to the dictator by collecting changes and submitting them over longer intervals. As a result, relying on identifiers to convey and contain versioning information limits the evolution of new and valuable methods of processing change in digital objects.

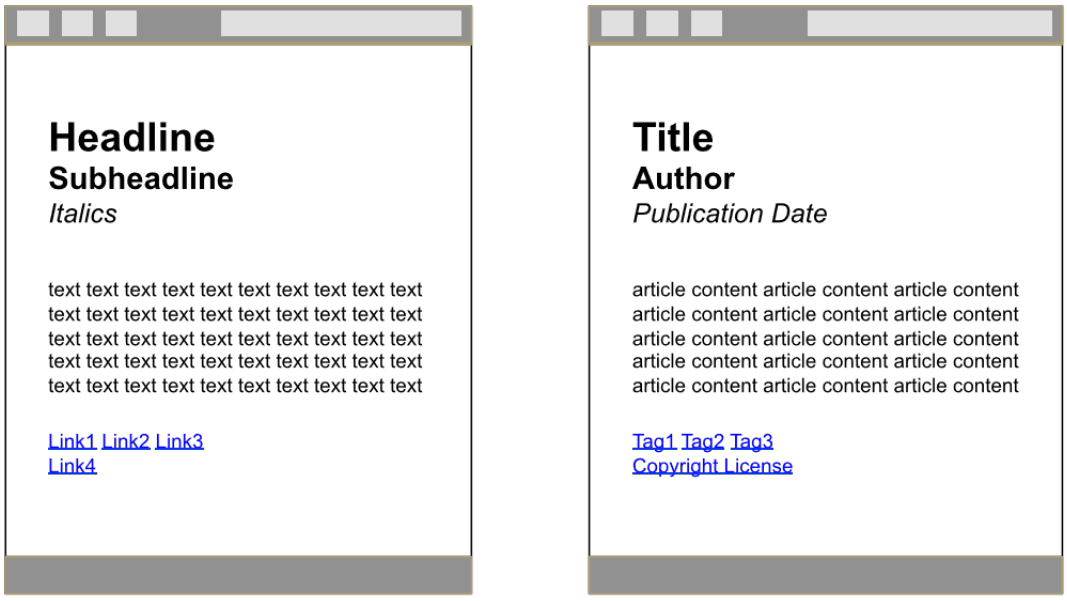


Figure 2.6: Illustration of the difference in what autonomous systems see when crawling a web page and what humans see when reading the same material. Figure 1 from [77]

2.5 Structured Data

The Resource Description Framework in Attributes (RDFa) framework encodes linked data vocabularies into HTML documents, and provides an opportunity to make change logs machine interpretable. [76]. Figure 2.6 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 based on location and style. RDFa seeks to encode that understanding natively for effective machine consumption. Extending this approach into publishing change logs, will allow linked data to capture the metaphorical meat of change content.

The implementation requires changing publishing practices from plain-text documents to something structured-data compatible such as HTML. The change also has the added benefit of making the logs available on-line, and thus, more

openly accessible to data users through the utilization of web based search engines. Large companies such as Google have already begun equipping their web crawlers to consume structured data such as RDFa from web pages. RDFa has already had significant success in adoption across a variety of web publication platforms and eases the search for their content [78]. The design of RDFa focuses on describing the web page's content through markup [77]. The underlying or resulting versioning data model may not conform with the format of content presented in the change log. Poor affinity would lead to a poorly structured graph or missing content, undermining the value gained by encoding linked data into the change log. As a result, another method using JavaScript Object Notation for Linked Data (JSON-LD) was pursued since its purpose is to store data separate from visible content.

The JSON data format allows web pages to store data for JavaScript applications within the document. It utilizes a simple and robust syntax to accommodate a wide variety of content. JSON-LD extends the original specification by defining rules which allow entries to resolve as web vocabularies, giving them a meaningful context [79]. Because it stores data separate from visible content, JSON-LD does not need to adhere with the constraints of visible content. Every linked data triple must instead be explicitly defined, meaning that resulting documents may likely be much larger than their RDFa counterparts.

2.6 Change Distance

A major function of versions is to communicate the amount of change which exists between two versions. The quantity plays a major role in determining the freshness of data within a collection, indicating its pertinence to new projects [80]. Additionally, changing versions are often used to signal other applications downstream that a new version may be necessary to adopt data improvements [81]. Many efforts currently to compute a distance measure relies on data provenance. Formalizing operations on provenance remains an active field of research [82]. Other approaches relate to determining semantic similarity in trying to summarize the data set and computing a distance measure [83].

2.6.1 Provenance Distance

Previous endeavors to extract insight into data set performance or behavior using provenance have provided exciting results [84]. The research, however, generally studies the current state of an object's provenance rather than compare two provenance graphs. As stated previously, versions result from slight variations between the provenance of two objects. The connection suggests that studying the variations' magnitudes will help predict the change's impact. The measurement known as provenance distance seeks to determine the impact of changes in provenance on new data versions through measuring graph edit distances.

The first ingredient necessary to calculate provenance distance is a linked data graph capturing the sequence of events leading to the old and new objects' creation, like the one shown in Figure 2.6.1. The graph shows the multiple lower level products involved in creating a Level 3 ozone indicator. This can be accomplished through the use of previously mentioned provenance models, but these graphs are not widely available. 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 similarity measurement. When calculating the similarity measurement of two graphs, algorithms determine how far the graphs are from being isomorphic [85]. Node labeling simplifies the similarity measurement process by providing nodes which must match together, and greatly reduces the complexity from computing generalized graphs. Graph Edit Distance, counting the edits necessary to transform one graph into another, provides a quantitative measure to associate with this process [86]. Some variations count edge changes [87].

In Figure 2.8, 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 an alteration in dependencies between objects used to generate a final notable product. Isolating changes responsible for differences in provenance can become difficult in complex environments as Tilmes observes in 2011,

> 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 hasn't changed, the data files in the whole process

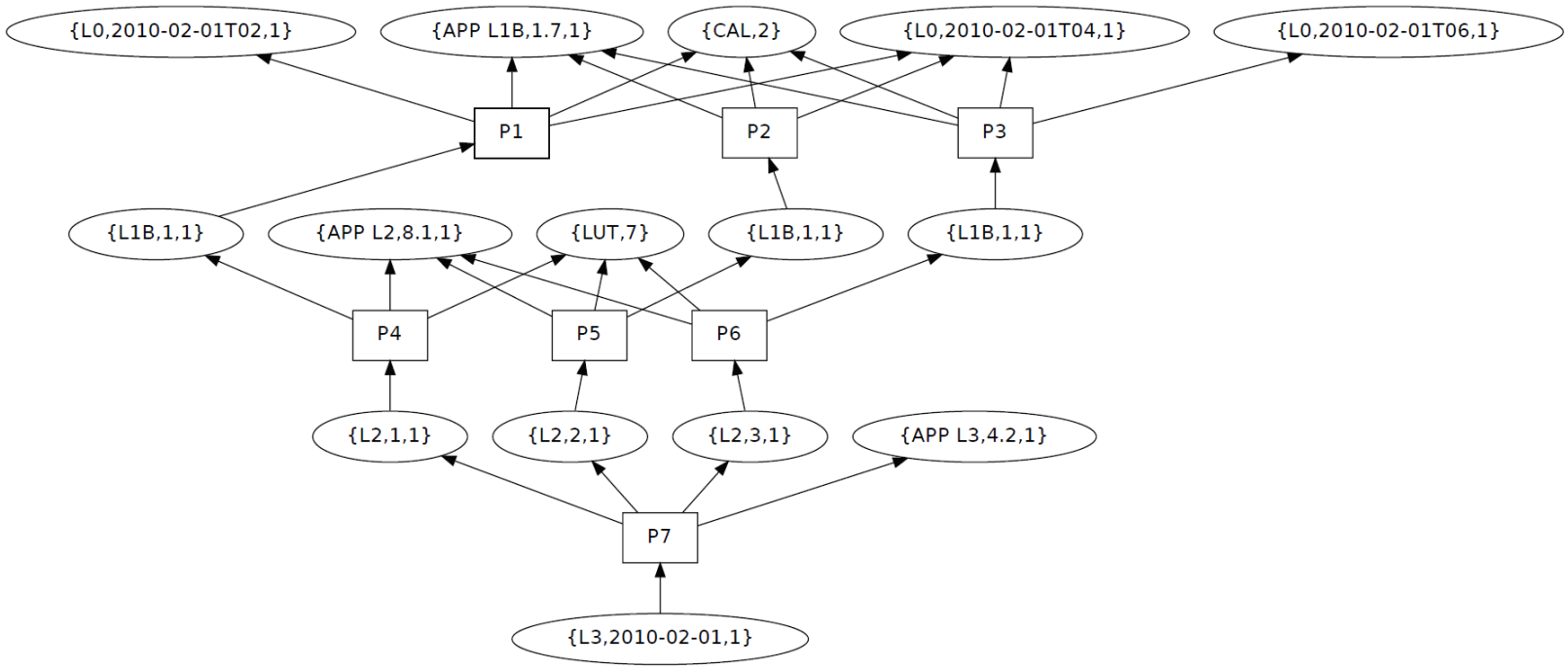


Figure 2.7: Provenance graph of a Level 3 data product, showing the inter-relations between different data products in generating the final product. Figure 2 from [81]

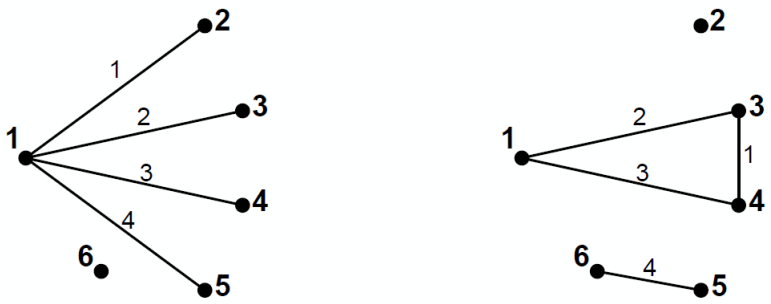


Figure 2.8: The labeled graph on the left transforms into the right graph under two edge edits. Figure 2 from [87]

have been affected by the change in the calibration.

[81]. L-number is shorthand for the level system featured in Figure 1.1. While provenance distance may be straight-forward to calculate, the indicator hides many insights into an object's behavior.

Methods to provide quality of service boundaries leveraging provenance already exist which compare workflows based on performance criteria [88]. These procedures focus primarily on quick retrieval and efficient storage instead of capitalizing on the latent information accessed by reasoning across data set versions [89]. Using only provenance data is insufficient to give insight into a change's impact because it does not provide information on structural or content differences which is what change logs provide. Measuring a change's impact with accuracy comparable to a change log requires a more detailed understanding and description than provenance can provide [73]. Sufficiently precise versioning measurements cannot be provided by provenance distance, but it could indicate the confidence of versioning results, which is out of scope for this project.

2.7 Summary

In order to better formalize data versioning information, an approach must be developed leveraging common aspects of very disparate versioning systems. A data model based around versioning operations instead of impact remains largely untouched across the field. Version identifiers must additionally be untangled from communicating change distance which change logs accomplish with greater detail.

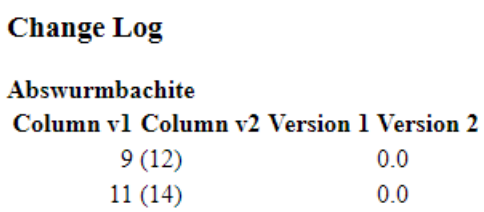
The logs, in turn, need to be extended for machines to consume, easing adoption as data set size grows through automation. Change measures utilizing version graphs rather than provenance graphs are also under-explored. Chapter 3.3 presents a model to create a versioning graph.

CHAPTER 3
MACHINE-READABLE CHANGE LOG

3.1 Introduction

Change logs explain the differences between versions; however, they are often only available in human-readable formats. Readability puts a limit on the length and extent of the log since a human will need to write it. Manageable change descriptions become difficult with large data sets featuring many changes, or data sets that change often, but these are exactly the data sets which need change logs the most. Automating the process will allow more data sets to provide change documentation in a timely fashion for data sets. Encoding the change logs with structured data will provide a means for users to efficiently consume change information. The additional encoding will inflate the size of a standard change log which becomes an issue with the change logs.

Change logs were generated for two data sets, the "Global Database on ^{3}He/-^{4}He in on-shore free-circulated subsurface fluids" data set and the Paragenetic Mode for Copper Minerals database. Following the practices of other change logs, the documents present before and after values for comparison which can be seen in Figure 3.1. The change logs identify challenges to adopting thorough change logs as a practice in versioning data sets.



Change Log

Abswurmbachite

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

Figure 3.1: Abswurmbachite entry in the Copper Dataset Change Log

Table 3.1: Files in the Noble Gas data set.

Filename	File Size (Bytes)	Rows	Columns	Total Cells
DB_HE_6733.xlsx	2682683	6733	199	1339867
DB_final-55-7262_2015_03_08.xlsx	2729060	7265	54	392310
NG_DB_final_2017_07_01.xlsx	4216595	8231	54	444474

3.2 Utilized Data Sets

3.2.1 Noble Gas Data set

The "Global Database on $^3He/^4He$ in on-shore free-circulated subsurface fluids" is a tumultuous database [90]. The first version, published in June 11, 2013, contains the information from 8 regions of the world united into a single file with around 199 columns. The next version of the database, published March 8, 2015, reduces the number of columns to 54, marking a drastic change. In addition, several columns changed the units with which they reported measurements. While usage documentation, explaining the content and use of the data, accompanied each version, no records were included indicating what changed between versions. A change log would be valuable guide with such drastic structural and content changes. The third and most recent publication came in July 11, 2017, with no changes to the number of files or columns, but many new rows. The structural summary of each of the files can be found in Table 3.1.

3.2.2 Copper Data set

The Paragenetic Mode for Copper Minerals database became available through collaboration with the author's lab to create new methods of visualizing mineralogy relationships [91]. The first version was collected June 8, 2016, with the update following soon after on August 8, 2016. Major edits are fairly limited with only 16 column additions and 2 removals between the versions. Value formats remain consistent from one version to the next, resulting in a much more condensed body of changes, making transitions more easily verifiable. Compared to the Noble Gas data set, it provides a more stable data platform to implement the versioning model in Section 3.3. The data from this work is also more processing friendly, making it

Table 3.2: Files in the Copper data set.

Filename	File Size (Bytes)	Rows	Columns	Total Cells
ParageneticModeTable_Cu_6.8.2016.xlsx	339175	705	37	26085
ParageneticModeTable_Cu_8.21.2016.xlsx	233715	685	51	34935

agreeable to automatic change log generation. An interesting thing to note in Table 3.2 is that the second version takes up less storage space even though it has more data.

3.3 Version Model Specification

When making change logs more approachable for usage in data sets, two approaches are available. The first approach continues writing change logs in only human readable language and relying on advances in natural language processing to allow computers to read the change logs. The second approach uses linked data to encode the change log with machine-computable statements. Since natural language processing is currently not sufficiently articulate, the second approach is taken. In doing so, a versioning data model needs to be developed which can capture changes in the way a change log organizes information.

A versioning data model needs to address a variety of needs not met by provenance models. In PROV-O and PAV, the modeled entities are exclusively one-dimensional with each version leading sequentially to the next one. The HCLS model, Figure 1.2, and Barkstrom model, Figure 1.3, however, display a more complex two-dimensional hierarchy. The tree models better capture the tiered granularity separating different versions which can result from a higher-tier macro change. These models also tightly couple new objects with changes to their underlying attributes. The tiered approach more clearly explains the scale on which two objects within the tree differ.

Provenance models provide concepts to sequentially order data objects but lack the ability to convey differences between farther spanning objects. In Figure 1.3, the left-most leaf node and the right-most leaf node differ by three changes

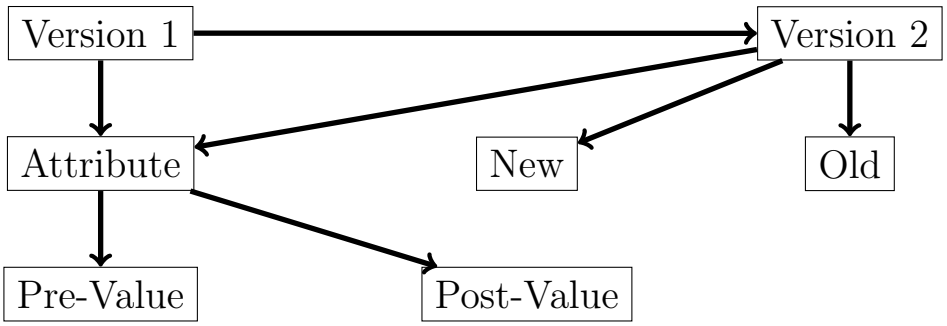


Figure 3.2: Provenance oriented versioning model.

at the data product level. A provenance model would need to rely on qualified properties to connect further annotations and describe the higher level changes. Remember that a common function of versioning systems is to provide a method to determine the amount of change or difference between two objects of a work. Much of the differences become lost when compressed into a single relation in a provenance graph. Additional annotations are often in natural language and do not provide a regular attribute to quantify.

The provenance models, on the other hand, do a much better job in explicitly defining the connection between objects which the tree models imply with structure. The versioning model must contain a mechanism to convey how changes to parts of an object contribute to that object's transition into a new version. The fundamental operations—**add**, **invalidate**, and **modify**—are used by the model to capture change in a more detailed manner. These details provide a mechanism to measure change between versions with better clarity than current methods.

3.3.1 Initial Approaches

The first approach, seen in Figure 3.2, simply extends the provenance relation with additional concepts to capture more types of relationships. Until the introduction of or comparison with Version 2, none of the concepts in Version 1 can be considered new or old. As the responsible party for introducing changes, Version 2 becomes associated with *New*, *Old*, and *modified* attributes. Version 1 also relates to *modified* attributes since it provides the *pre-value* used to contextualize Version 2's *post-value*. The pre and post values are included so that a user can see how much

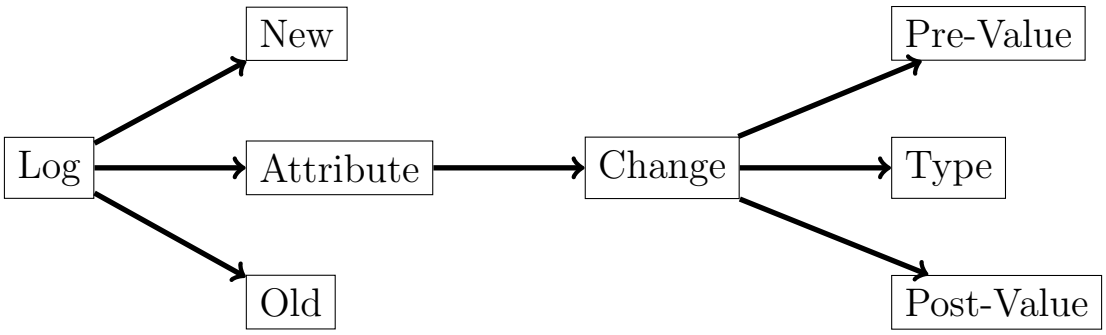


Figure 3.3: Change log based versioning model.

the attribute has changed, much like with a change log.

Adding the attributes as concepts to the model addresses PROV's and PAV's flat approach to version relations, but the attributes do not capture the inter-relation between objects for *New* and *Old* attributes. Having Version 2 be responsible for all the changes causes issues with the model since it must be associated with attributes from an entirely different object. The *Old* attributes should not exists within Version 2. Associating *Old* attributes with Version 1 would be more appropriate and intuitive to understand. The model does not capture the **type** of change, making the result a listing of attributes without the version differences to contextualize the relationship between the versions.

From a different direction, Figure 3.3 shows a model created by starting from the change log. Attributes are attached to the log as the primary indicators for old, new, and modified concepts. Change logs often break down and group changes by attributes. For modified attributes, an additional change concept is associated, encapsulating the values and nature of the change. At the far right side of the figure is a concept called *Type* which indicates more specifically the nature of the change, for example unit of speed to another. Pre- and post- values are also included to explicitly define the change concept.

The primary drawback of the log-based construction is that the change log assumes all responsibility for every modification to the data even though the document only reports the differences. The version objects are also left out of the model, leaving the log concept in possession of the attributes. One of the major breakthroughs with this model construction is that while specific values are kept in the

log, those values do not need to be in the model. By encoding the type of change, the need for actual values becomes superfluous as change type is more generalizable across domains and contexts.

Figure 3.4 combines the provenance and change log approaches by capturing the transition from Version 1 to Version 2 in the change log. The idea here is to enable distance capture between versions by encapsulating all changes within the log concept. The changes are then associated with specific attributes. Pre- and post-values do not appear in the model because the presence and kind of change is more valuable than assessing the explicit values involved. As a data set becomes more volatile, more values would need to be stored, resulting in more of a copy of the data rather than a summarization of the changes. Notice in Figure 3.4 that *Attribute* has now become disconnected from either version. Reconnecting the *Attribute* concept brings into question which version it should be associated with since it exists in both. The larger issue with both the log-based and hybrid approach is that the model resembles a tree more than a graph, making linked data queries less powerful, as most of the concepts are disconnected.

A fourth formulation, in Figure 3.5, leverages the insight that when a change interacts with an attribute, the attribute is different in the next version. The model addresses the attribution problem by forming two attributes, each associated with a different version. These attributes inform a change which acts upon both version concepts. The *Log* object is dropped for the model since it is a method to convey change and not an actor involved in the change. From the highly connected construction, new and old *attributes* no longer need to be explicitly stated, but they can

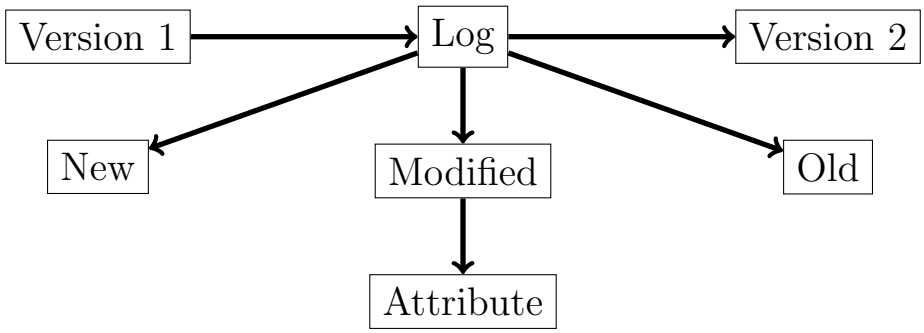


Figure 3.4: Hybrid provenance and change log versioning model.

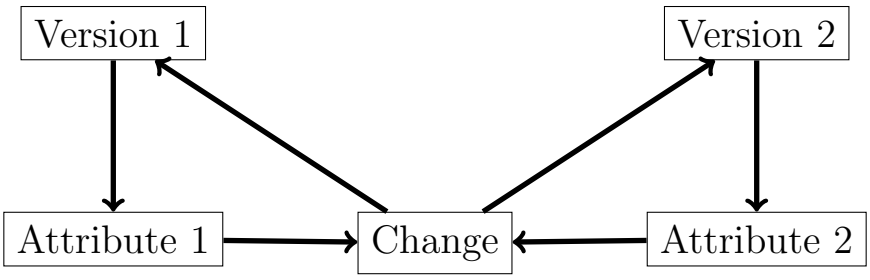


Figure 3.5: Highly connected model of just versions, changes, and attributes

be implied from the model's structure. A new attribute would not exist in *Version 1* so *Attribute 1* and its associated properties (arrows) are removed, leaving a unique construction implying an attribute addition.

One observation is that the relation from changes to versions is redundant since the links from *version* to *attribute* to *change* implies the same relationship. Removing the explicit relation would shorten the number of triples required to encode a change and improve scalability. The versioning graph using this highly connected model would also be easier to query if the edges were oriented in the same direction, additionally implying that change flows from one version to the next. These final observations result in the current versioning model.

3.3.2 Model Objects

The versioning model incorporates three kinds of objects: **versions**, **attributes**, and **changes**. A **version** object represents the items being compared such as a book or spreadsheet. In PROV, a **version** would likely correspond with the *prov:Entity* involved in a *prov:wasRevisionOf* property. The **attribute** object refers to specific parts which make up a **version**. **Attributes** could be lines in a book or columns in a spreadsheet. Including **attributes** addresses the lack of detail involved in a *prov:wasRevisionOf* or *pav:previousVersion*. The relationship between **versions** and **attributes** captures the influence that changes in the underlying part will have on the overarching **version**. Because the model refers to specific parts of a **version**, the **version** concept corresponds most closely with a FRBR **manifestation** rather than an **expression**. The presence or absence of an **attribute** is used to determine the kind of **change** which occurs to the **attribute** between **versions**. **Changes**

are used to link together **attributes** from different **versions**. The **change** captures a difference between the old **version** state and the new **version** state. While the **change** object greatly resembles a PROV qualified property, its form can change depending on the kind of **change**, like a *schema:UpdateAction*.

3.3.2.1 Left-hand Right-hand Convention

In the following diagrams and figures, the original or base version and its attributes will be placed on the left-hand side and the new version will be placed on the right-hand side with its attributes. References to the versions as previous and next are avoided since sequencing may not play a major role in distinguishing versions. Scientific data in large repositories often track sequential releases of data, but a book may have different versions distinguished by printed language. To recognize this distinction, objects will be referred to as the **left-hand version** or **left-hand attribute** when they are not sequentially or temporally related.

3.3.3 How Changes are Represented in the Model

The model bases **changes** around the three core versioning operations because their commonality across systems provides a fundamental basis for comparisons. **Additions** occur when an **attribute** appears only in the **right-hand version**. When an **attribute** only shows up in the **left-hand version**, the model captures this as an **invalidation**. Finally, a **modification** change has **attributes** in both the **left-** and **right-hand versions**, but it only connects two **attributes** if their values are different. These three combinations cover the possible situations within the model.

3.3.3.1 Modification

The **modification** relation occurs when an **attribute** appears in both **versions** and their values are different. In Figure 3.6, a **modification** is captured between two versions. 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.

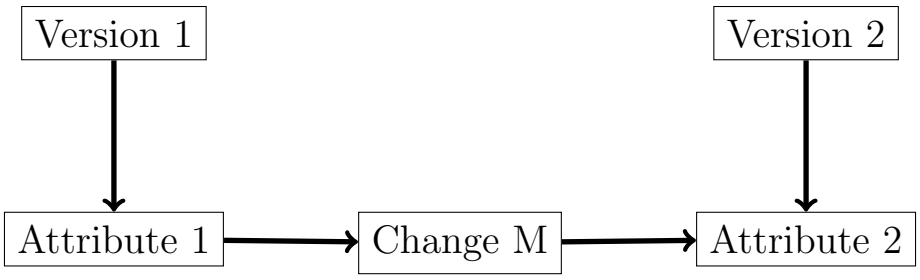


Figure 3.6: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

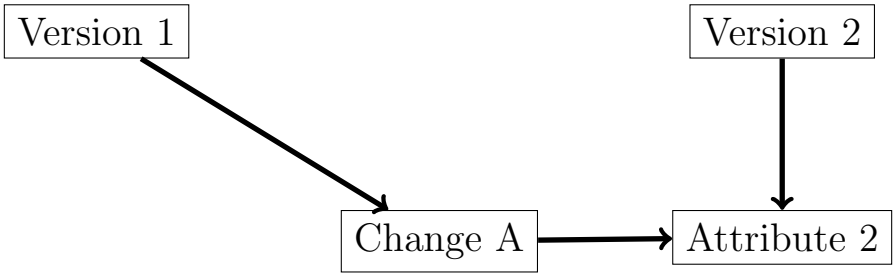


Figure 3.7: Model of the relationships between Versions 1 and 2 when adding an Attribute 2 to Version 2 as a result of Change A

The specific values pertaining to Attribute 1 and Attribute 2 are not captured by the model because acknowledging that a difference exists is more important. Extending the model to properly communicate the significance of a modification for a wide variety of domains would require sizable domain knowledge and would be outside the scope of this project. In addition, the model would essentially begin storing a copy of the data set, leading to space and redundancy concerns.

3.3.3.2 Addition

In Figure 3.7, the **addition** model differs from the **modification** construction by the absence of Attribute 1. The absence creates a disconnect between "Version 1" and "Change A". We know that a connected graph will be desirable to accommodate traversal using linked data query languages so "Version 1" must be reconnected to the other concepts in the model. A **property** is used to create a path between the two **attributes** to indicate the contribution of "Version 1" to the change's lineage. The path does not show that "Version 1" informs or creates "Attribute 2", while that

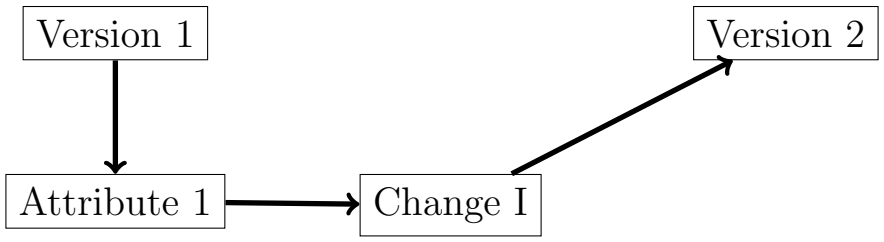


Figure 3.8: Model of the relationships between Versions 1 and 2 when invalidating Attribute 1 from Version 1 as a result of Change I

may be true. The construction was also chosen to create a symmetric orientation with the **invalidation** change.

3.3.3.3 Invalidation

The *invalidation* model has a missing **attribute** on the right-hand side of the relation, contrary to the **addition** construction. As a result of the invalidation, an attribute no longer exists in the **right-hand version**. As seen in Figure 3.8, the invalidation change concept matches to the Version 2 object. Just like in the **addition** model, this construction maintains a link between the two **version** objects. In this case, it makes more conceptual sense, however, because "Version 2" invalidates "Attribute 1" by omitting it.

3.4 Encoding a Change Log

Very little natural language is used in the change log to regularize the format and improve compatibility with RDFa. The change logs follow a common format with three sections: Additions, Invalidations, and then Modifications. The sections may be further grouped by column or row additions. The division means that changes are not published into the change log as they are found, but instead organized and grouped beforehand.

Employing RDFa means that the document must be written using HTML formatting. Listing 3.4 shows the text necessary to layout the first four lines of Figure 3.1. While the content only shows four lines, the underlying markup takes up three and a half times as many lines. Line 2 of Listing 3.4 states that all following resources will be **attributes** of Version 1. Line 3 defines such an **attribute**. Lines 5

through 8 define the changes Abswurmbachite undergoes. Because RDFa allows the statements to be embedded within the content, the triples can appear along with the text they describe. Lines 11 and 12 define complete triples which do not appear in the visible document. The lines complete the graph, but must be included in spans because RDFa only allows a single triple within each tag. Modifying the tags' order so that the spans are unnecessary would cause the visible content to appear in a confusing and disorganized order, rendering the document machine-readable but not human-readable.

```
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 1: Abswurmbachite RDFa

After encountering the limitations of using RDFa to include the versioning graph into the change log, JSON-LD was used. The JSON data does not need to conform to the structure or ordering of visible content in the change log. Listing 3.4 provides the alternative encoding of the Abswurmbachite entry from RDFa. The entry is significantly longer, almost three times longer than the RDFa entry and ten times longer than the original visible content. Instead of including all the data

in the beginning or end of the document, each change block is separated into the particular *div* section for that change. This choice allows consumers to extract pertinent change information without needing to ingest the entire versioning graph.

The change logs created with RDFa or JSON-LD demonstrates progress towards documents which are both human and machine-readable. The implementation provides evidence that JSON-LD is better suited to embed a versioning graph into a change log than RDFa. RDFa suffers limitations since it is constrained by the content's structure. The **modify** relation presented in Figure 4.1 is unbalanced and the right-hand side of "ChangeCAM00111" links only to the column **attribute** but not to the corresponding row **attribute**. This stems 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, defeating the purpose of using RDFa to implicitly encode the versioning graph into the document.

Both structured data implementations break up the graph across **attributes** so that individual parts of the graph can be extracted. The practice of a one-node JSON object is generally helpful for many web applications to load data quickly, but since the change log is not an application, it makes more sense to break up the content. Changes to individual **attributes** can be identified using anchors on the web page, then agents need only extract and parse the linked data to these specific entries. This way, a subgraph of only the pertinent attributes can be created without first ingesting the entire versioning graph.

An unexpected challenge with the change logs is the larger file size and difficulties in loading the Noble Gas data set's JSON-LD change log. The problem results from needing ten lines to express a single row in the change log. Noble Gas also had an impressive number of **modifications**, some of which are shared across all rows in the data set. Repeated modifications over rows would account for the explosion in entries within the change log.

3.5 Cold Land Processes Field Experiment

The NASA's Cold Land Processes Field Experiment collects ice measurements from a wide number of sites in the arctic [92]. The data was published originally in ASCII, but the data was later ported into shapefiles as well. A known issue with the transition was that comments in the Summary data set were left out do to format limitations. A change log was created to determine whether additional differences existed within the two formats.

The initial problem with forming a change log is that rows in the data did not include unique identifiers to align attributes within the versions. A unique tuple, using values from 5 columns of the data, was created to match rows together since the order of the rows did not correspond between the two formats. An immediate problem that results was some differences occurred within the columns necessary to form the unique tuple, namely some values disagreed by 0.5. Following the correction, the data could be correctly aligned.

The change log found that for columns "soil sample A" (SL_A) and "soil sample B" (SL_B), many entries consistently disagreed between versions. Disagreements resulted from values being recorded with 'y' or 'n' for a number of values but with numbers for the rest. Since the values in the shapefiles are stored in a database format, an column must conform to a pre-specified type, causing the character encodings to appear as 0s in the shapefiles. Additionally, the columns experienced 0.5 misalignments and null values of -999.0 being encoded as 0. The columns are encoded as integers, causing the disagreement.

With further investigation, misalignment in the values caused repeated inquiries into differences between expected and reported values. A passage in Parsons et al. [93] states,

> "After soliciting feedback from the CLP Working Group and working closely with several of the CLP investigators, we settled on a set of four simple, fixed-width, ASCII tables for each IOP. This was simple and readily understandable and, we thought, flexible enough to be used by most applications. Unfortunately, we found that, although these ASCII tables worked well for many applications, it was not very easy to import

> them into Excel and ArcGIS, two of the most common data analysis tools. To ensure future readability of the data, we were committed to using a textual file protocol (Raymond, 2004). Yet, we also sought to provide the data in a format that would facilitate analysis and examination. Therefore, we modified the fixed-width tables to conform to the Microsoft comma-separated value (CSV) file convention while keeping fixed-width columns. This facilitated import into Excel and other spreadsheets. The text files are the primary archival format of the data, but we also created ESRI shapefiles for GIS users. Documentation clearly describes both CSV and shapefile versions of the data, and indicates that the only differences in content are in the free-text comments recorded by the surveyors (shapefile text fields suffer from a short field width, so surveyor free-text comments are only included in the CSV files). By working closely with current data users while considering future users and applications, we were able to develop a data format that met current needs, is flexible enough to accommodate future applications, and is simple enough to survive long-term technological changes."

The solution resulting from the differences was to indicate that the preferred and official version of the data was the ASCII representation. The deference to a different format, unfortunately partially undermines the benefits of increased accessibility provided by also providing the data in shapefile format.

3.6 Change Log Analysis

With a trade-off of 14 HTML lines for every visible line and 40 HTML code lines to each visible line, space utilization is a very present concern. Table 3.3 shows the size of each encoding of the change log as well as the percentage in size as compared to either of the files involved in the version transition. 'Text' denotes the encoding control where no structured data is included into the change log. Alone, the control is already double the size of either file. The RDFa encoding more than 20 times the size of the original files, exceeding the size of the control by more than 10 fold. A separate file was generated in turtle format to observe whether taking just

Table 3.3: Noble Gas change log size: 1st Transition

Encoding Type	File Size (Bytes)	% of File 1	% of File 2
Text	5575405	207.8294	204.2976
RDFa	62175478	2317.660	2278.2745
Turtle	80919783	3016.375	2965.1156
JSON-LD	130134071	4850.893	4768.4577

Table 3.4: Noble Gas change log size: 2nd Transition

Encoding Type	File Size (Bytes)	% of File 2	% of File 3
Text	403227	14.7753	9.56286
RDFa	4168390	152.7409	98.85678
Turtle	4515435	165.4575	107.0872
JSON-LD	8095372	296.6359	191.9884

the linked data values would reduce the information to a more manageable size, but the turtle file was still 30 times the size of the original files. Adopting the versioning model and encoding it into a change log will very likely require significant storage investment.

Table 3.4 shows the change log sizes for the second version transition in the Noble Gas data set. Notice that the second transition has much smaller text encodings compared to the original files. The RDFa and JSON-LD encodings once again 10 and 15 times, respectively, the size of the text encoding. The turtle encoding, however, is smaller than the RDFa encoding, but still 16 times the size of the original file. Looking at Table 3.5, the second transition had 20 times fewer **Modify** entries, leading to a much smaller turtle file.

Another way to evaluate the performance of the change log is to look at the number of change entries compared to the number of changed values, in the Copper database's case spreadsheet cells. From Table 3.6, the behavior of the encodings is very similar to the second transition of Noble Gas. The text format is smaller than the original data set, but the encoded files are at least 10 times the size of the

Table 3.5: Noble Gas Turtle files

Filename	Add	Invalidate	Modify	Total Triples
changelog.ttl	608	216	102830	110602
changelog2_3.ttl	990	24	5369	8146

Table 3.6: Copper change log size: 1st Transition

Encoding Type	File Size (Bytes)	% of File 1	% of File 2
Text	140131	41.3152	59.9580
RDFa	2032823	599.343	869.787
Turtle	1538772	453.680	658.396
JSON-LD	3500067	1031.93	1497.57

Table 3.7: Changes to Copper Data

Change Type	Rows	Columns	Cells Affected
Add	1	16	10995
Invalidate	21	2	2145
Modify	NA	NA	2628

database files. To determine the number of cells affected by a change, the number of cells added by new rows is summed with the number of cells added by new columns, using the width and length of file 2. The cells affected by removals is based on the length and width of the first file. The number of remaining cells, equivalent between the two files is 23940. Since **Modifications** are reported cell-by-cell, the number of cells affected is equal to the number of **Modifies**, 2628. The rows and columns that **Modify** affects are not available because the changes appear inconsistently across the rows and columns meaning a reported value would be misleading. The complete counts are reported in Table 3.7.

The triples used to explain changes as a percentage of the cells affected is reported in Table 3.8. Smaller percentages indicate a higher efficiency of each triple since one triple can explain changes to multiple cells. Notice that **Adds** are much more efficient in explaining changes than **Invalidates** due to the kind of change each triple explains. **Invalidates** explained changes to rows primarily while **Adds** mostly explained changes to columns, but since columns are much longer, **Adds** ended up scoring higher on efficiency. **Modify** triples are extremely inefficient and also

Table 3.8: Change capture efficiency in Copper Data

Change Type	Triples	% of Cells Affected
Add	17	0.065%
Invalidate	23	1.1%
Modify	2628	100%

account for more than a majority of the changes to the data, meaning that **Modify** triples most likely account for the bloat in the physical representation of the triples. Not represented in the change log are the unmodified cells which account for 89.02% of the matching cells between the Copper files. The analysis indicates that while **Add** and **Invalidate** may be very efficient in expressing changes, improvements to encoding and **Modify** capture efficiency are needed to bring down the storage costs of automated change logs. The bloated change log size likely explains the dearth of data set change logs in practice since using the storage space for more data would be more valuable.

3.7 Summary

The automated change log generation yielded some unexpected results. Automated change logs standardize the process to capture change within a data set. While more popular text-only change logs could be adopted, a versioning data model was necessary to make the logs also machine computable. The computability improves user navigation over large data sets. The drawback is that the encoded change logs are reliably much larger than the original data set in bytes. The storage space cost likely contributes to the reason that change logs are often unseen in data set documentation. The automation and inclusion of change logs inform consumers how much the data set has changed.

The versioning model provides a method to capture change information in greater detail than current provenance models. The inclusion of **versions** and **attributes** into the model connect changing items with the objects they influence. The **changes** create a ladder-like structure to connect **version** objects in greater detail. Each rung of the ladder can not only be counted, but also grouped into types of change according to the respective operation. The method of instantiating a versioning graph will be covered in Chapter 4.

The human-readable presentation defines the structure which tags in the change log must take since maintaining human-readability is desired. The structure then determines the order in which linked data statements must appear in the log to encode the graph with RDFa. The ordering creates limitations on how strictly the

encoded graph adheres to the specification from Chapter 3.3. While construction of the change log is automated, encoding through RDFa significantly reduces the source HTML readability. In other applications using RDFa, the triples describe and link the text encapsulated by HTML tags. The versioning graph exclusively ignores the marked up content and links together tags or explicitly defines full triples in span tags.

Change logs are much less restricted when encoded using JSON-LD rather than RDFa. The encoding format pulls the graph out of the attributes where they do not interact with content and into a separate script section. The method causes a drastic expansion per change in necessary text. The decision to divide up JSON-LD objects by the row in the change log they describe likely contributes significantly to the overhead necessary for the encoding. The division was made with the forethought that change log consumers may desire to only ingest specific subgraphs of the versioning graph. Separated JSON-LD objects will likely need to be merged in the future to save space for data sets with many changes.

The resulting logs end up very large and sometimes do not load in a browser. Reassuringly, both data sets displayed the same space usage complexity with RDFa being ten times the plain text size and JSON-LD twice the size of a change long in RDFa. The relationship unfortunately means that a JSON-LD change log, with more readable source code, is twenty times the size of a plain text change log. The Copper data set's logs were reasonably responsive, displaying in seconds, but the Noble Gas's change log did not. In order to retain usability, there will need to be methods optimizing change log structure or representation.

```
<h3>Change Log</h3>
<div about="v1:Abswurmbachite">
<span style="font-weight:bold"
 ↪  property="http://www.w3.org/2000/01/rdf-schema#label">Abswurmbachite</span>
<table>
<tr  id="ModifyChangeAbswurmbachite12">
<td align="right"> 9</td>
<td >(12)</td>
<td>          </td>
<td>       0.0</td>
<script type="application/ld+json">
[
{
"@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld",
"@id": "http://CUdb.com/v1/AttributeAbswurmbachite9",
"@reverse": {
"hasAttribute": "Version1"
},
"@type": "vo:Attribute",
"label": "Primary",
"undergoes":
 ↪  "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/CUjsonlog.html#ModifyChangeAbs
},
{
"@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld",
"@id":
 ↪  "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/CUjsonlog.html#ModifyChangeAbs
"@type": "vo:ModifyChange",
"resultsIn": "http://CUdb.com/v2/AttributeAbswurmbachite12"
},
{
"@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld",
"@id": "http://CUdb.com/v2/AttributeAbswurmbachite12",
"@reverse": {
"hasAttribute": "Version2"
},
"@type": "vo:Attribute",
"label": "Primary"
}
]
</script>
</tr>
</table></div><br>
```

Listing 2: Abswurmbachite JSON-LD

CHAPTER 4
CHANGE METRICS

4.1 Introduction

Machine computable change logs provides a very powerful means to begin answering basic versioning questions in a formal and systematic manner. From the change log, a linked data versioning graph can be extracted and the changes counted to communicate how different are two versions. The data model was constructed to allow a wide variety of ways to connect together versions such that more complex analytics could be performed using the versioning graphs. The analytics show that producers must be very transparent when communicating the methods data producers use to assess change impacts as shown in Section 4.4.2.

When versioning a data set, researchers very rarely ask whether two objects can be compared. The data producer often establishes the context in which data objects are sufficiently similar—to use terms from FRBR—**expressions** of the same **work**. Confirming the context prior to making version comparisons is fundamental to ensuring that the resulting versioning graph contains meaningful results. The data sets described in the following section have sufficient context as established by their producers. Using the data in these data sets, the model from Chapter 3.3 is instantiated into versioning graphs. The graphs are encoded into HTML change logs using RDFa and JSON-LD. These graphs allow for an analysis of the change between versions, which gives insight into the version identifier. Finally, a version graph is used to classify the kinds of change separating versions of a data set to determine the utility.

4.2 Implementing the Versioning Model

The following subsections detail the steps used to implement a versioning graph using the model defined in Chapter 3.3 and the challenges encountered. Section 4.2.1 goes through the decisions made to align the attributes within the Noble Gas dataset and within the Copper data set. The alignments create a formula to detect changes

and assign them to either an **add**, **invalidate**, or **modify** change. A change log can then use the assignments to organize a presentation of the change data. The underlying versioning graph exists as linked data encoded within the change log, but can also appear as explicit linked data statements. The linked data uses a custom-made versioning ontology (VersOn) to express the data model using the *vo:* namespace. The procedure within this section defines the process used to create versioning graphs found in all the following sections of this Chapter.

4.2.1 Form a Mapping

A mapping specifies the method to determine the **attributes** of a versioning graph and how to compare them. For spreadsheets and table-based data, row and column indexes initially seem an ideal attribute, but edits often show the contrary. The Noble Gas data set needed a mapping to align the spreadsheet's columns since 140 columns were removed from the first version. The remaining columns in the second version no longer had the same column indexes that they did in version 1 so the column headers were used instead. The Copper data set retains many of the original columns, but their ordering has changed between versions. In addition, rows must be aligned since both a row and column attribute are necessary to uniquely identify a cell. The Noble Gas data set split up its rows across eight files, each file representing a separate region of the Earth. Instead of forming eight versioning graphs or having eight left-hand versions, the files were collected together into a single abstract collection which is then mapped to the right-hand version. Creating eight versioning graphs would also form eight separate change logs which doesn't make sense since each file forms only a part of the entire work and the second version collects all entries into a single file. Multiple left-hand versions also doesn't make sense since this creates one change log and graph, but the files are no longer associated with each other. Cells need to be uniquely identified since this is where a comparison will be made to determine whether a **modify** change has occurred in a spreadsheet.

Once aligned, determining which attributes have been added, invalidated, or modified is straight-forward. Attributes which only exist in the original or left-hand

version have been invalidated. More specifically, a set of attributes $\mathcal{I} = \mathcal{R}_l - \mathcal{R}_r$ where $\mathcal{R}_l$ and $\mathcal{R}_r$ correspond to the row identifiers of the left-hand and right-hand versions, respectively. Likewise, a set of attributes $\mathcal{A} = \mathcal{R}_r - \mathcal{R}_l$ contain all the added attributes. Performing the same operations on the columns result in sets of the added and invalidated columns. A script then iterates over the remaining cells which exist in both versions to determine if they differ, resulting in a **modify** change. The unchanged cells form a set of entries which do not appear in a change log or the versioning graph. The attributes in these sets are then minted into URIs and linked together into the versioning graph, or they can be used to publish a change log.

4.2.2 Generate Versioning Graph

The versioning graphs presented in this section were created by extracting triples from the associated change log which will be covered in Chapter 3. The statements making up the graph could have alternately been published by writing out the triples directly instead of encoding them into a change log. Figure 4.1 displays a subgraph of the Noble Gas data set's versioning graph between versions 1 and 2, highlighting each of the three change operations. Notice how the versioning graph differs from the provenance graph in Figure 4.2. The versioning graph unpacks the *prov:wasRevisionOf* relationship into explicit components. These components reveal more detailed differences between version 1 and 2 of CAM001 in the provenance graph which are the differing compilation activities. The change log encoded the triples in RDFa, resulting in the attribute "AttributeCAM00111v2" to the right of the **modify** change. Because RDFa does not naturally support multiple predicates while also conforming to the content structure of the change log, an attribute was created to combine both the row and column identifier for the changing cell. Separating the attributes would require multiple dedicated HTML tags which don't appear along with content. Including these tags would diverge from benefits of encoding triples as attributes. Figure 4.1 also shows that even though many columns are added when a new row is added, the row identifier can be used to summarize the columns additions.

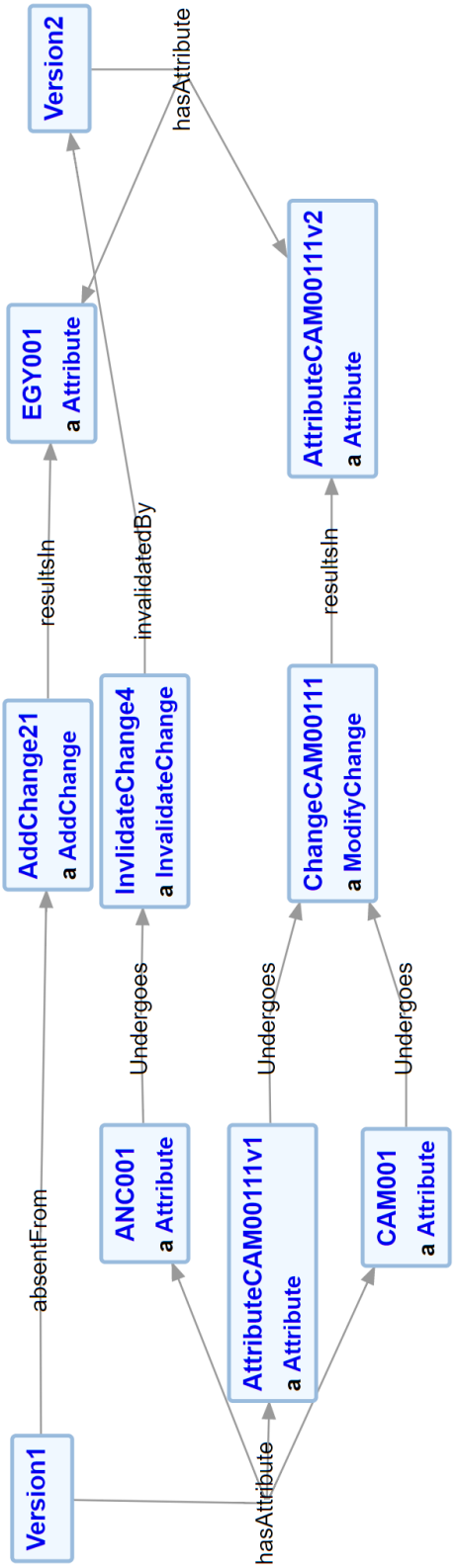


Figure 4.1: Some initial entries from versions 1 and 2 of the Noble Gas data set

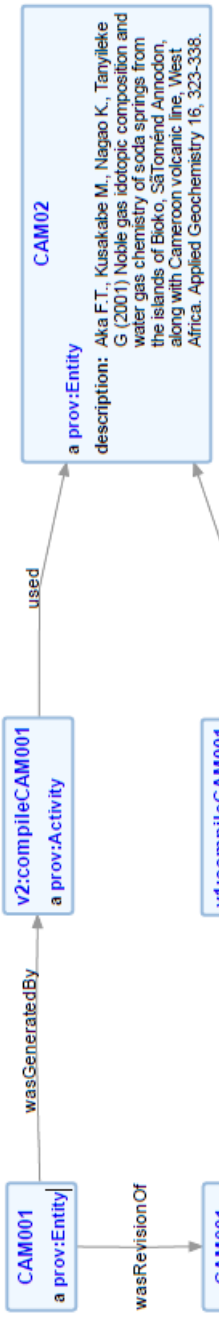


Figure 4.2: Provenance graph for the CAM001 entry of the Noble Gas Database. Other than the labels, the structure of each data object is very much the same.

Another modification to the implementation differs from the original versioning model. The **modify** construction defined in the model only covers the case where a single attribute is sufficient to define a change relation. The **modification** captured in spreadsheets describes a cell which requires a row and column identifier to indicate uniquely. The implementation demonstrates that using multiple attributes is an allowable, sometimes necessary, construction.

Listing 3 presents the statements in turtle format necessary to express that the entry EGY001 has been added to the data set from Version 1 to Version 2 as shown along the top of Figure 4.1. The namespace for many of the URIs is ⟨http://rdfa.info/play/⟩. RDFa allows identifiers to refer to an element on the web page, and the web tool which generated the triples from RDFa, therefore, used its URL as a namespace to produce a valid URI.

```
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 3: Noble Gas Add in Turtle

Figure 4.3 shows a similar subgraph from the Copper data set versioning graph. The graph was assembled using an RDFa change log and also displays a merged attribute on the right side of the **modify** change. In the full versioning graph, multiple of each change is present, forming a zipper or ladder-like structure. As a result, each **add**, **invalidate**, or **modify** change is given separate names for each instantiation.

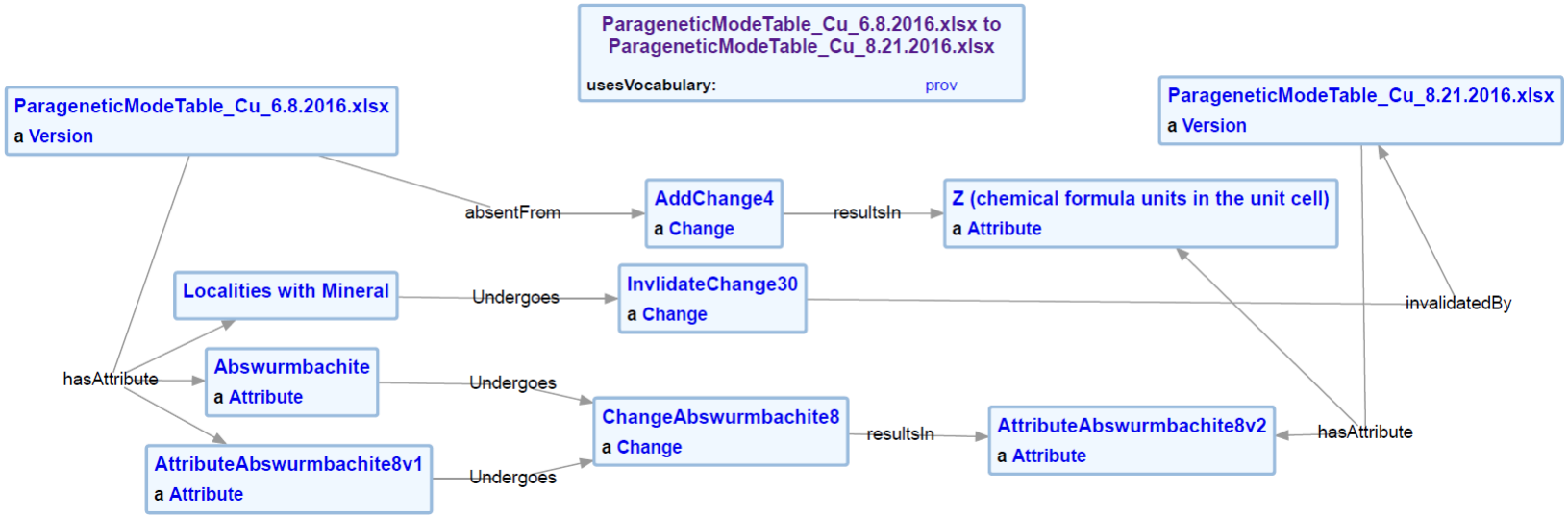


Figure 4.3: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications.

4.2.3 Graphs with Multiple Versions

Figures 4.1 and 4.3 depict a comparison between only two versions, but a project can contain more than two objects. Case in point, a third version of the Noble Gas data set was released on July 11, 2017. Figure 4.4 shows a subgraph that contains changes from all 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 data set. This entry then undergoes a modification change in columns 29, 31, and 43 when comparing versions two and three. Entry TUR030 goes through a modification change in column 11 from version one to version two. The entire row, however, becomes invalidated in version three.

Notice the difference in how Figure 4.1 and Figure 4.4 refer to columns. Figure 4.1 used linked data extracted from a change log employing RDFa, forcing the row identifier and the column identifier into the same concept. The way nesting works in RDFa means that ChangeCAM00111 cannot back reference multiple concepts in a single statement, therefore AttributeCAM00111v2 was used to imply CAM001. Figure 4.4 used linked data extracted from a JSON-LD encoded change log. Since the log can use explicit statements, the column identifier refers to the entire column and can be used to identify changes in the same column across multiple rows.

4.3 Change Metric

Use Case 2 addresses the use of versions to communicate how different two objects are. Many versioning systems use dot-decimal identifiers to signify whether a change is large, medium, or small. The exact requirements to determine change size differs widely across different domains and applications. The versioning graph provides a new, more regular method to quantify change between objects using versioning operations. The work done with GCMD Keywords shows the qualitative relationship between version identifiers and change distance. Work with the MBVL data set then extends VersOn to give more detailed accounting with the change capture method.

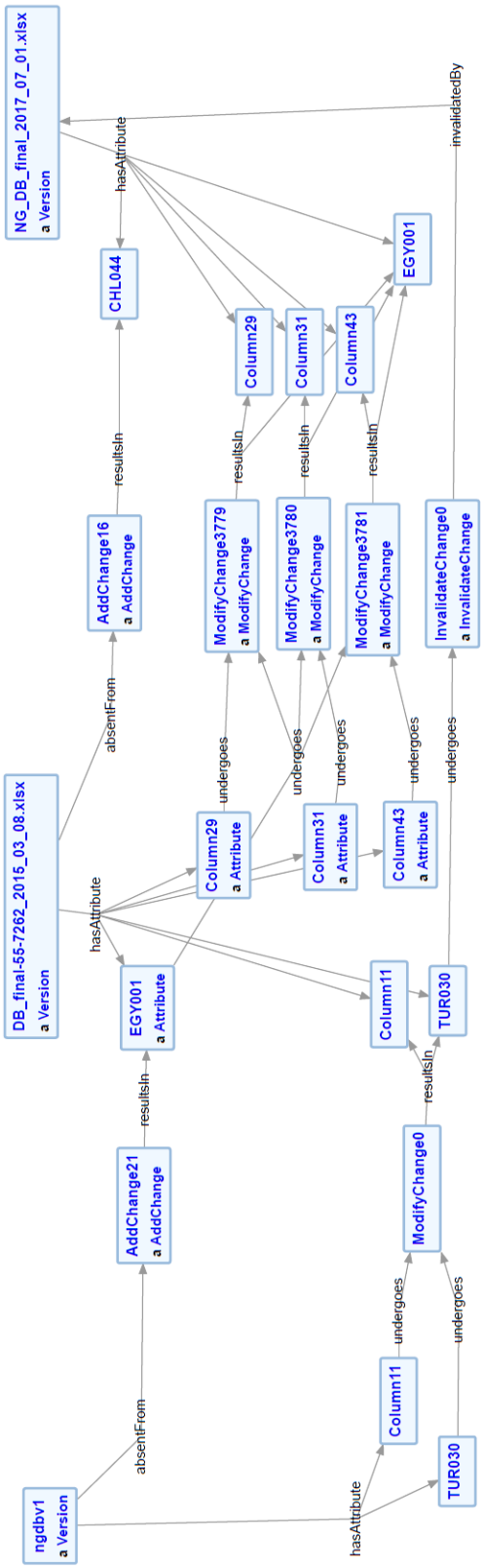


Figure 4.4: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications after the publication of the third version.

4.3.1 Utilized Data Sets

4.3.1.1 Global Change Master Directory Keywords

The Global Change Master Directory (GCMD) is a metadata repository used by NASA to store records of its available data sets [94]. They employ a set of keywords to make NASA Earth Science data sets searchable. These words tag and label datasets into strictly defined categories [95]. GCMD Keywords do not qualify as a standard web ontology since GCMD Keywords do not constitute a class hierarchy. The management team stored early versions of the keywords in Excel spreadsheets, later using a centralized distribution system, but data is not available prior to June 12, 2012. The Key Management Service now serves the keywords directly in a variety of formats. Each version of the keywords, encoded in RDF, was downloaded into separate files. Only versions from June 12, 2012 and after were available, resulting in 9 version files. Each keyword corresponds to a unique identifier, and when combined with a web namespace, resolves to a data description of the keyword. Every identifier can be referred to per version by including the version's number at the web identifier's end, meaning that identifiers are consistent across versions. The taxonomy 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 [96]. The tree's root is the keyword, "Science Keywords." The data set provides an interesting study case due its long sequence of versions and ready use of linked data technology [97].

4.3.1.2 Marine Biodiversity Virtual Laboratory Classifications

The Marine Biodiversity Virtual Laboratory (MBVL), based at Woods Hole Oceanographic Institution, provides data and services for the study of marine biology with an integrative approach [98]. In the application studied, a choice of algorithm and taxonomy pairings must be tested on a known population in order to estimate their performance with an unknown microbial population. The original sequences belong only to the species listed in Table 4.1. The original population's census is not available to the author, and only the list of species is known, forming the first data set in this section. These sequences are then grouped and classified by a

Table 4.1: List of species in the original population.

Acinetobacter baumannii	Actinomyces odontolyticus	Bacillus cereus
Bacteroides vulgatus	Clostridium beijerinckii	Deinococcus radiodurans
Enterococcus faecalis	Escherichia coli	Helicobacter pylori
Lactobacillus gasseri	Listeria monocytogenes	Neisseria meningitidis
Porphyromonas gingivalis	Propionibacterium acnes	Pseudomonas aeruginosa
Rhodobacter sphaeroides	Staphylococcus aureus	Staphylococcus epidermidis
Streptococcus agalactiae	Streptococcus mutans	Streptococcus pneumoniae

specific taxonomy and algorithm pairing. The workflow utilizes two taxonomies, the Ribosomal Database Project (RDP) and the Silva taxonomy. Using these databases, the SPecies-level IdentificatioN of metaGenOmic amplicons (SPINGO) or the Global Alignment for Sequence Taxonomy (GAST) algorithms assign taxonomic ranks to each sequence. The process produces four data sets, each using the same grouping identifiers and having the same size in each group. Since the data sets have the same number of sequences, the primary differences between the data sets are the ranks assigned to each sequence.

4.4 Global Change Master Directory

4.4.1 Global Change Master Directory Versioning Graph

The Global Change Master Directory establishes the context that each **manifestation** of their keyword list is related versions. Since the unique identifier for each keyword remains the same across versions, the unique keyword identifier can be used to align a mapping across versions. **Additions** and **invalidations** are detected by checking an identifier's presence within both versions. A **modification** occurs when a keyword's *skos:Broader* property differs between adjacent versions. The alignment assumes that there is not reason a keyword's preferred label would change, but still reports a value when it has new entries in the "notes" property. A difference indicates that the word has been moved to a different place within the taxonomy since identifiers do not change across versions and a keyword only has one parent concept. Changes over consecutive versions can be collected into a single graph using the method in Section 4.2.3 to chain together versioning graphs. A change log was generated for each pair of consecutive versions in GCMD Keywords

Table 4.2: Global Change Master Directory Keyword Change Counts

Transition	Add	Invalidate	Modify	Total
June 12, 2012 to 7.0	310	9	22	341
7.0 to 8.0	503	6	79	588
8.0 to 8.1	277	28	22	327
8.1 to 8.2	53	1	26	80
8.2 to 8.3	58	0	13	71
8.3 to 8.4	53	0	1	54
8.4 to 8.4.1	86	13	8	107

and embedded with JSON-LD. Versioning graphs for each adjacent version was created by extracting JSON-LD from the corresponding change log, and entering the triples into a Fuseki triple store.

4.4.2 Connecting Change Counts to Identifiers

The **add**, **invalidate**, and **modify** counts for each transition are presented in Figure 4.5. The query used to extract the counts is found in Listing 4. Notice the sharp spike in adds and invalidates when transitioning from version 8.4.1 to 8.5. The version identifiers indicate that at most a minor or technical change has occurred, but the counts of **addition** and **invalidation** changes in this transition is more than triple the counts in either of the previous **major** transitions. Not only should a small transition not produce changes of this quantity, but the data set's size is on the order magnitude 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. To provide context, NASA mandated a transition to secure protocols, and the group changed the name space to ensure the URIs remained resolvable. Since the identifiers are unique, the new name space means they no longer refer to the same object after the protocol change. Because the keyword identifiers no longer match, the mapping approach results in the total invalidation of keywords from 8.4.1 and the addition of keywords from 8.5. The dot decimal identifier for the transition from version 8.4.1 to 8.5 does not match the number of changes in the versioning graph.

Changing the mapping method to account for the new namespace provides

```
PREFIX vo:<http://orion.tw.rpi.edu/~blee/VersionOntology.owl>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>

SELECT ?p (COUNT (DISTINCT ?s) as ?count)
{
?s a ?p .
?p rdfs:subClassOf vo:Change .
} GROUP BY ?p
```

Listing 4: This query compiles the counts for each subclass of Change in a GCMD versioning graph.

a pathway to compare the perceived change by the producer as evidenced by the version identifier with the amount of change in the versioning graph. To do this, the mapping treats identifiers with HTTP and HTTPS the same. Differences in change magnitudes become much clearer after controlling for the altered name space in Figure 4.6. 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. The transition from version 8.4.1 to 8.5 also seems to follow this trend.

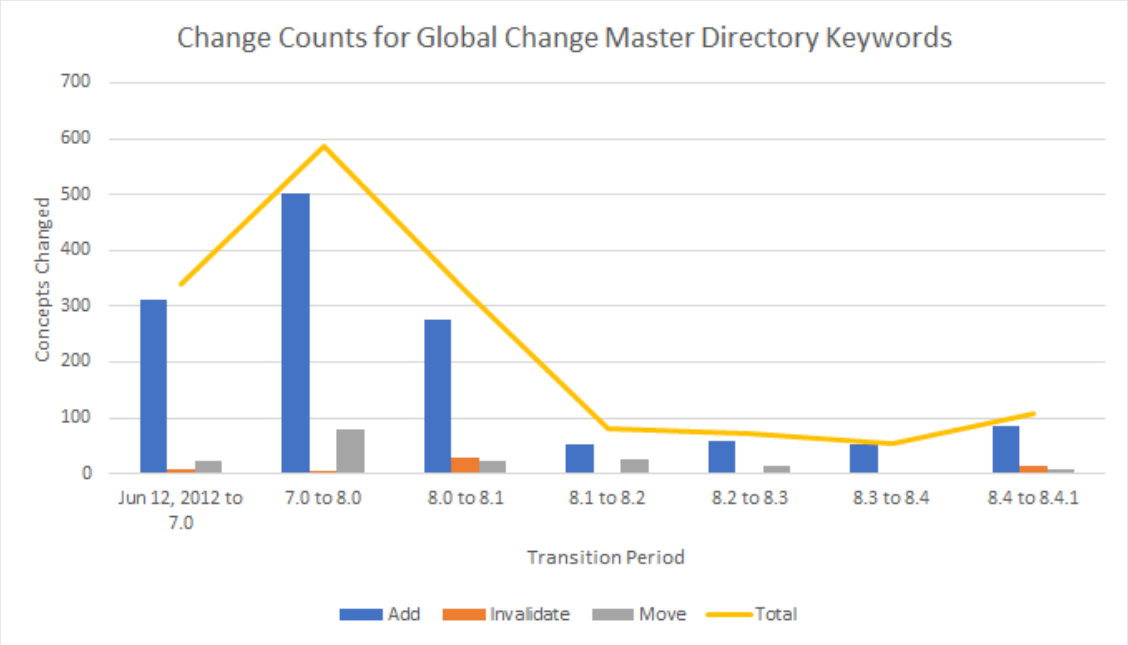


Figure 4.5: Add, Invalidate, and Modify counts from the beginning of the Keyword Management System to Version 8.4.1.

Table 4.3: Difference in Version 8.5 mapping methods

Mapping Method	Add	Invalidate	Move	Modify
Standard	3097	3031	0	0
Silent	68	2	22	0
Bridged	68	2	22	3007

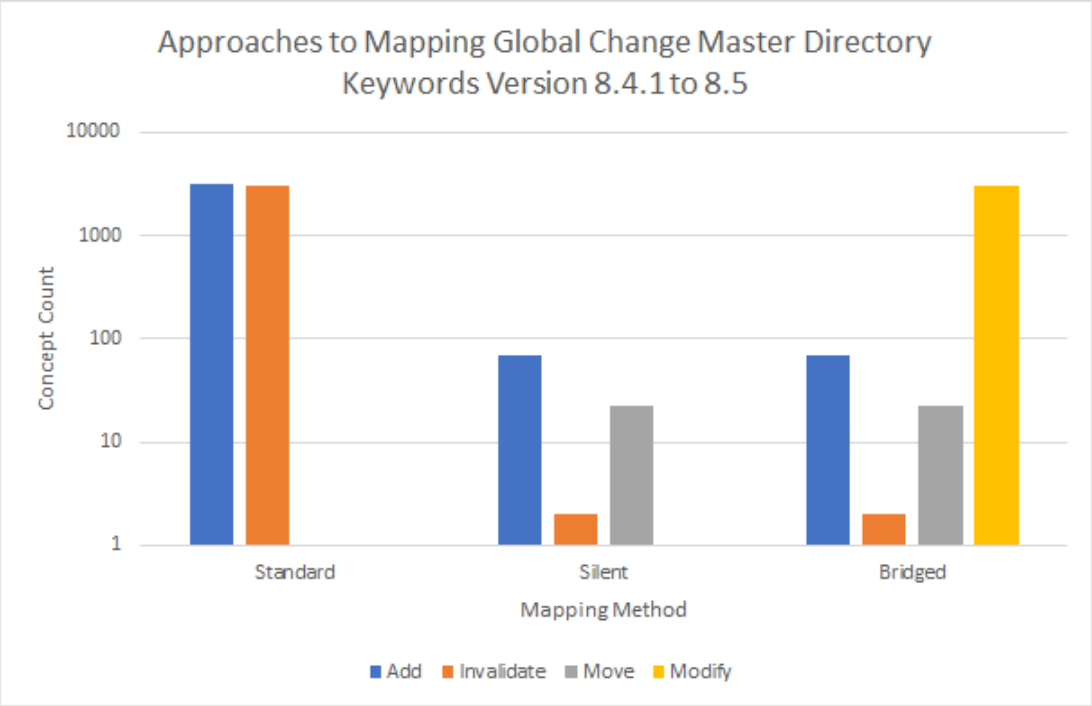


Figure 4.6: Add, Invalidate, and Modify counts using different methods of mapping identifiers in Global Change Master Directory Keywords Version 8.4.1 to 8.5.

The **additions** in "8.4 to 8.4.1" in Figure 4.6 numbers almost a hundred, providing evidence that the trend of decreasing order of magnitudes may now continue as the granularity of the version identifier increases.

4.5 Marine Biodiversity Virtual Laboratory

4.5.1 Variant Versioning Graph

The experiment conducts activity over two phases in this procedure. The first phase takes sequences from the original known population and feeds the sequences though a particular algorithm/taxonomy combination to produce a candidate classification. Since the classifications for the known population sequences are unavail-

able, there is not sufficient context to perform a valid comparison with the candidate classifications. The second phase compares the performances of each candidate classification of a algorithm/taxonomy pair. The use of **add**, **invalidate**, and **modify** varies slightly in this application since all the results use the same sequences. A versioning graph utilizing just the sequence identifiers would only result in **modify** changes when taxonomic ranks differ since the sequence identifier exists in both data sets. The mapping instead uses the sequence identifiers to align comparisons and then the taxonomic rank classification to determine the kind of change. If the right-hand result specifies more taxonomic ranks, the relationship is an **addition**. If the left-hand result is more specific, then the relationship is classified as an **invalidation**. If both results have the same precision but the name differs, then the link is a **modification**. Otherwise, no change is detected.

Figure 4.7 shows the changes detected when varying either the taxonomy or the classification algorithm. Only the taxonomy or only the classifier differs in each comparison to control for overlapping influences that having both a different taxonomy and classifier may introduce. Each bar indicates the total number of differences between sequences for a specific kind of change. The bars are further broken down by the taxonomic rank at which the difference occurred. For example, in "Silva vs RDP, Gast", a notable number of classifications differed at the species rank. The graph also indicates that using the RDP taxonomy often produces more precise classifications since both "Silva vs RDP, Spingo" and "Silva vs RDP, Gast" feature a larger number of **additions** than any other change. The classifier comparisons feature a high number of **invalidations**; however, "RDP, Spingo vs Gast" also displays a higher number of **modifications** than **invalidations**.

4.6 Version Graph Discussion

The versioning graph successfully addresses the concerns of Use Case 1 by capturing all the differences within the Noble Gas data set and within the Copper data set into a versioning graph. Some additional concerns had to be addressed, such as multiple files in a version and dual attribute identification, during the implementation of the versioning model. The multiple files in the first version of the Noble

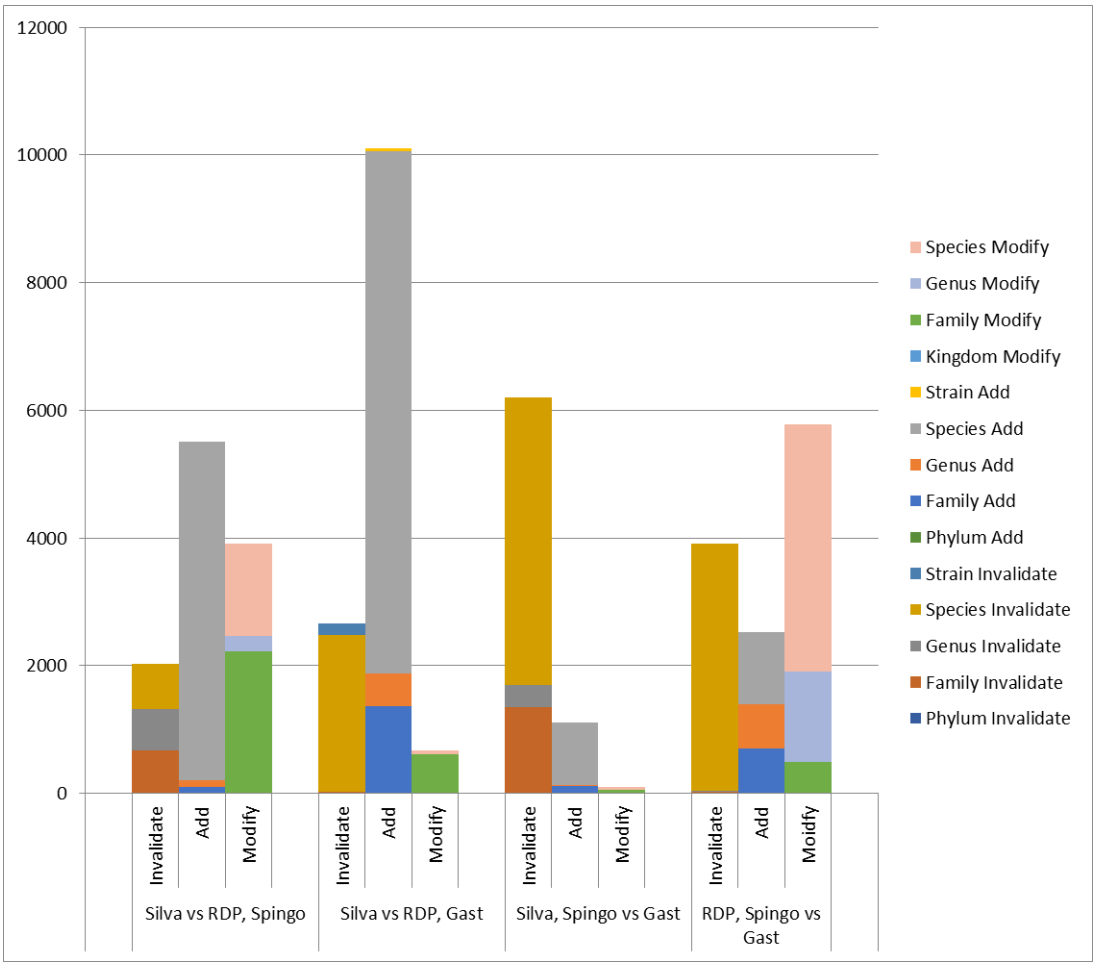


Figure 4.7: Compiled counts of adds, invalidates, and modifies grouped by taxonomic rank across algorithm and taxonomy combinations.

Gas data set needed to be collected into a single concept in order to preserve the one-to-one relation between versions. The grouping simplifies the graph structure as well as reduce the complexity of a change log encoding.

In Section 3.3, there is only one **attribute** on each side of the interaction. Figure 4.3, however, shows two **attributes** used to characterize the *vo:ModifyChange*. While the model only shows one **attribute**, it was found that in some applications, multiple **attributes** may be necessary to properly model a single change. The construction does not even need to have the same number **attributes** on both sides of the **change**. The flexibility becomes important when trying to model, for example, a single location entry being split into separate latitude and longitude entries.

The version graph's construction allows multiple versions to be linked together.

The graph provides not only greater continuity than Schema.org's properties, but also greater detail than PROV's versioning properties. Continuity is important since many versioning linked data alternatives view version change as a single contained **activity**. When linking together multiple versions using a versioning graph, the relationship between non-adjacent editions becomes 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.

The model struggles with discontinuous changes to an **attribute** across multiple versions. Since the model does not capture when an **attribute** doesn't change, it is possible for an **attribute** in an earlier **version** to become disconnected from later **versions** due to inactivity. For example, in Figure 4.4, 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 disconnected **attributes**.

4.6.1 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 would reflect an order of magnitude division among the change counts of the versioning graph. According to the *Keyword Governance and Community Guide Document* [99], "Full GCMD keywords list releases get a new major version number (e.g., 8.0). Incremental releases for updates to topics, terms, and variables get a new minor version number (e.g., 8.1). The change counts for Versions 'June 12, 2012' to 8.4.1 demonstrates a threshold of changes necessitating a full keywords list release. The document does not explain the purpose or distinguishing qualities of versions with a new revision number, 8.4.1. Version 8.5, however, was named with respect to perceived taxonomy changes and did not consider underlying linked data practice revisions. The conclusion can be obtained by looking at 4.6

and noticing that the only matching approach without a bar equal to the size of the entire data set is the 'Silent' approach. For a purely URI based comparison in 'Standard', Version 8.5 definitely falls under the category of full release since an entirely new list of words is released. Users using the 'Bridged' approach would also see a new full release because all old words have had the URIs edited. Version name assignment based on producer perception and not allowing users to assess change measures is concerning. Making sure that data consumers have the ability to assess change in data sets when the requirements for change differs between producer and consumer must be addressed.

The analysis does not to claim that change counts should be the sole mechanism in determining version identifiers. The counts, however, can provide a more quantitative method to compare version differences. In Figure 4.5, the yellow line indicates the total changes made to the data set, performing a similar function as the major/minor/revision version identifier. Breaking up the changes into types reveals additions dominate manipulations to the data set. Addition, invalidation, and modification provides deeper insight into how a data set is changing, but some changes can be more impactful than others which this model does not capture.

4.6.2 MBVL Analysis

In Section 4.3, the versioning process was used to compare the performance of different taxonomy and algorithm combinations. The data set diverges from many of the common understandings of versions since each of the versions are not sequential and are largely independent. The data set of species names in the initial population would not have produced very meaningful results if applied to the versioning model since it lacked sufficient data to map the other data sets together well.

In Figure 4.7, the first set of columns in 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 are 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 "uncultured bacterium". This analysis presents evidence that using the RDP taxonomy with the SPINGO algorithm will produce the most accurate classification results.

4.7 Summary

The results in this chapter implements the versioning model and demonstrates the process and challenges experienced in this endeavor. The entries in a data set is separated into groups of **additions**, **invalidations**, **modifications**, and unmodified by their attributes. The grouping occurs over multiple files in the first version of the Noble Gas data set, and the solution was to collect them into a single unit. The collection keeps the files as one unit, but does not end up addressing other approaches to multi-part versions.

These operational groups organizes the data into a form to publish into a versioning graph. The approach used to create the graph involves extracting the linked data from a marked up change log. The decision resulted in constrained representations of the versioning graph, resulting from demands of the encoding methods. Graphs created using freer form statements, such as the one in Figure 4.4, demonstrate an opportunity enable querying over different dimensions of the data. Changes for specific columns can be queries as easily as individual rows.

The ability to link changes of multiple versions together results as a side effect of the model construction. Continuously linked changes opens up avenues of exploration to follow change as it propagates through versions. While change logs

will provide a more focused comparison, a triple store with a multi-version graph would give a view of the work through time. Considering the Noble Gas data set's versioning graph's size, many versions may be difficult to store with large, volatile data sets.

The MBVL data set demonstrates a case where versioning graphs can be used to compare the performance of different taxonomy/algorithm pairings. The ability derives from sub-classing each of the add, invalidate, and modify changes to give a better perspective where the pairings differed. This approach of extending the versioning graph adds domain knowledge to the version comparison and helps contextualize the observed differences.

CHAPTER 5
Data Volatility

5.1 Introduction

Capturing change counts begins to provide evidence for avoiding to use only versions to manage and describe data set change. Figure 4.5 only shows the relation of the changes to each version but disconnected from time. The changes were applied to each version, but the versions are not set to a strict schedule, meaning that many changes few changes could be applied to a version over a long or short period of time. The change rate communicates expectations for the data consumer on the value of a data set since committing to a data set that will soon be invalidated is problematic. The following chapter will look into data volatility, the likelihood or rate of data change, and look at how versions can hid the actual change rate of data sets.

5.2 Determining Volatility

Instead of charting the version changes in evenly wide bars, the versions are spread across time based on the time of publication to the KMS as seen in Figure 5.1. Since each of the versions were dominated by the **Add** counts, the count is divided by the number of days between the publication of a version on the left side of the line and the release of the replacement version on the right side of the line. The height of the line on the chart gives the steady rate of change until the release of the new version. The area underneath the line is the total amount of change the new version introduces. Since each version packages together all the changes into a single release, the actual change rate is unknown.

Three observable clusters appear in the time aware presentation of the versions, highlighted in Figure 5.2. According to the *Keyword Governance and Community Guide Document* [99], "Full GCMD keywords list releases get a new major version number (e.g., 8.0). Incremental releases for updates to topics, terms, and variables get a new minor version number (e.g., 8.1). The statement explains the activity in

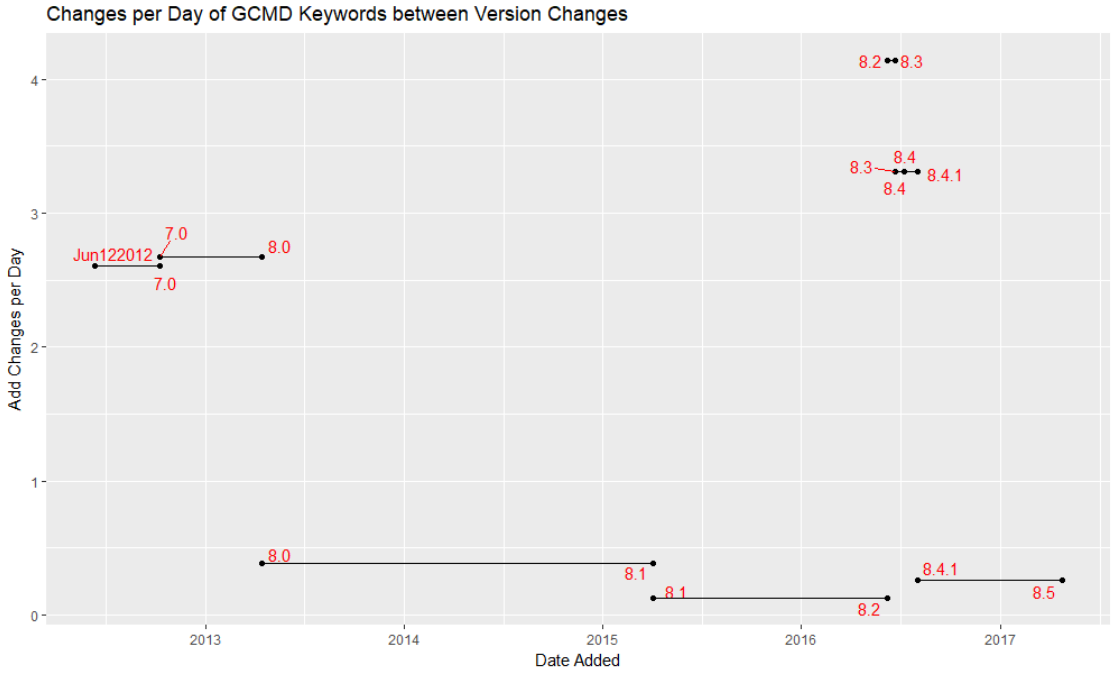


Figure 5.1: Add counts for all versions of GCMD up to 8.5 evenly distributed over the time of version validity.

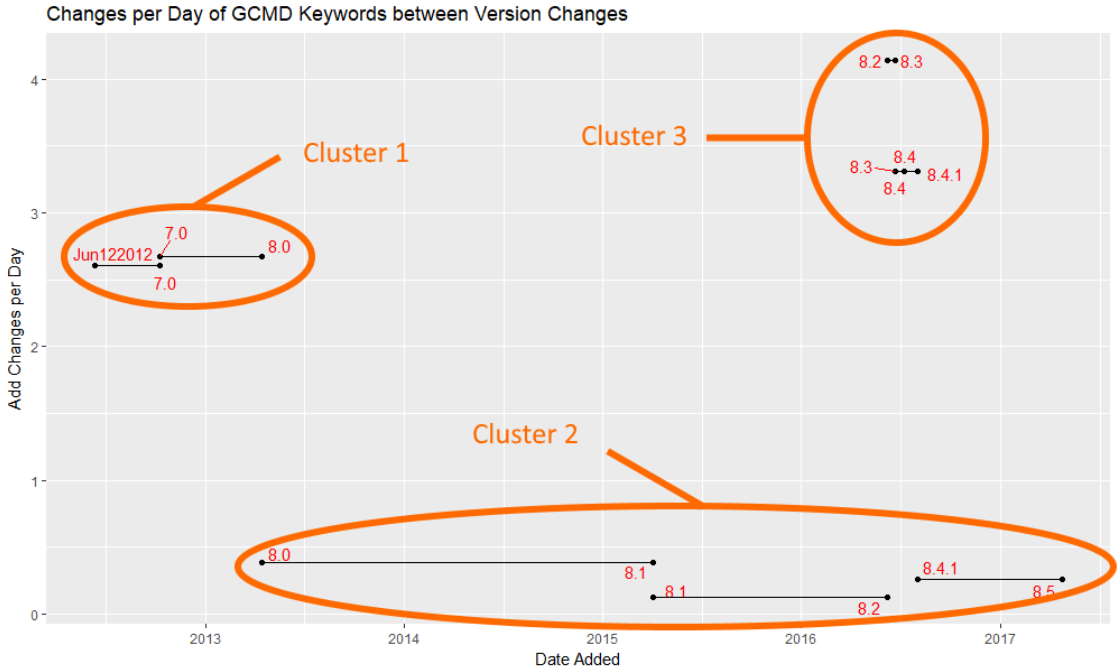


Figure 5.2: The change rate of different versions organize into three visible clusters. Cluster 2 denotes a sudden burst of version releases which is notable.

Table 5.1: Global Change Master Directory versions with old start time changes.

Version Name	Publish Date	2008	2014	2015
8.2	June 7, 2016	0	4	2
8.3	June 21, 2016	0	7	1
8.4	July 7, 2016	5	0	1

Cluster 1 where there are sufficient changes to warrant a full release of the keywords. Cluster 2 captures the change rate and duration of minor versions, except those from 8.2 to 8.4.1 which are in Cluster 3. Cluster 3 demonstrates a flurry of activity occurring between June 7, 2016, to August 2, 2016. Considering the previous pattern of taking at least six months between releases, three minor version releases within as many months is highly unusual.

An immediate concern is that Cluster 3 does not result from a sudden burst of activity, necessitating rapid version replacement. An inquiry into reasoning behind the successive publication returned a statement that the government customer had requested the action. Another way to dig into the behavior is to look into the impact assessments accompanying the versions. Impact assessments prior to Version 8.5 are not publicly available, and only assessments for versions 8.2, 8.3, and 8.4 were received upon request. Of the 6 requests affecting Earth Science Keywords in 8.2, published June 7, 2016, 4 were made in 2014, and the remaining 2 were made in 2015. Version 8.3 had 8 entries in its impact assessment with 7 entries originating in 2014, and the remaining entry from 2015. The 6 entries 8.4s impact assessment has 5 entries from 2008 and 1 entry from 2015. The data is collected in Table 5.1.

5.3 Earth Observing Laboratory

The Earth Observing Laboratory (EOL) of the National Center for Atmospheric Research (NCAR) distributes small data sets, around 10-12 files per data set, regarding lower atmospheric data beginning in 2005 [100]. The EOL data sets are somewhat unique in the data set size means management often does not require automation. In mid-2014, EOL began assigning versions to stored data sets. When receiving a new version of a data set from a researcher, the practice is to upload the

Table 5.2: Version Content of Earth Observing Laboratory Data Sets

Number of Versions	Number of Data Sets
1	1155
2	141
3	26
4	10
5	3
Total	1335

entire new data set, and replace all old files.

Of the 1335 data sets maintained by EOL with versions, only 180 data sets had more than one version. The full distribution of version counts is in Table 5.2 The 1155 other data sets were filtered out since change counts could not be computed for single-version collections. Since all the files are replaced on an update and a unique file identifier like a hash sum was unavailable, file matching between versions rely on filenames to perform change mappings. For all files that matched names across versions, the relation was classified as **Modify**. The approach will over-count the number of modifications, but provides an upper bound on the data set volatility in the repository. Each count is then normalized by the number of files in the previous version to standardize comparison between data sets regardless of data set size. The average for each data set is taken for each change type.

5.4 EOL Versioning Behavior

Given that EOL replaces the entire old data set when updating, the expected behavior of the transitions would be **Modifies** concentrating close to 1 and **Adds** and **Invalidates** distributed close to 0. The assumption is that researchers have little reason to change the file naming scheme. The data surprisingly indicates that data sets in EOL primarily gravitate towards **Addition** and **Invalidation** values of 1. **Modify** counts score more close to 0 in a complete reversal of expectations.

Figure 5.3 shows the distribution of **Add** scores. The primary feature of the chart is the bar situated in the '[0.9-1' range, meaning that about 45 data sets add a number of files equal to the original size of the data set. Secondary features include the bars on the far right and far left of the chart, but the bar on the right side is

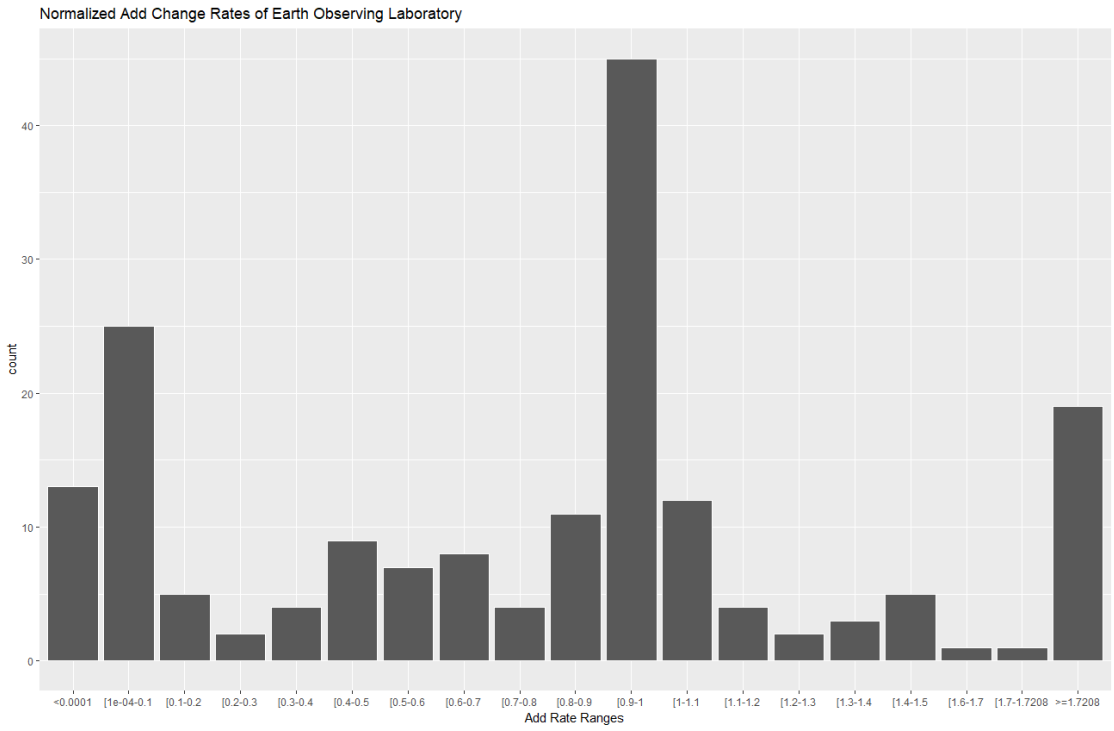


Figure 5.3: Distribution of average normalized Add counts for each data set in Eath Observing Laboratory.

Table 5.3: Normalized Change Statistics

Stat	Add	Invalidate	Modify
Mean	0.714312707	0.654819294	0.345180706
Std. Dev	0.509878564	0.420093557	0.420093557
Min	0	0	0
Q1	0.28635075	0.142857	0
Med	0.9146635	0.9642855	0.0357145
Q3	1.00358625	1	0.857143
Max	54.25	1	1
IQR	0.7172355	0.857143	0.857143

a collection of outliers. In the outlier data sets, the size of the data set increased drastically compared to the behavior of other data sets managed by EOL. Outliers are determined by collecting values above 1.5 times the interquartile range (IQR) showing in Table 5.3. A more muted distribution appears around the 0.5 mark where data sets grow more gradually.

The normalized **Invalidation** score in Figure 5.4 shows a majority of data sets removing all or almost all files in the data set. Coupled with the information that

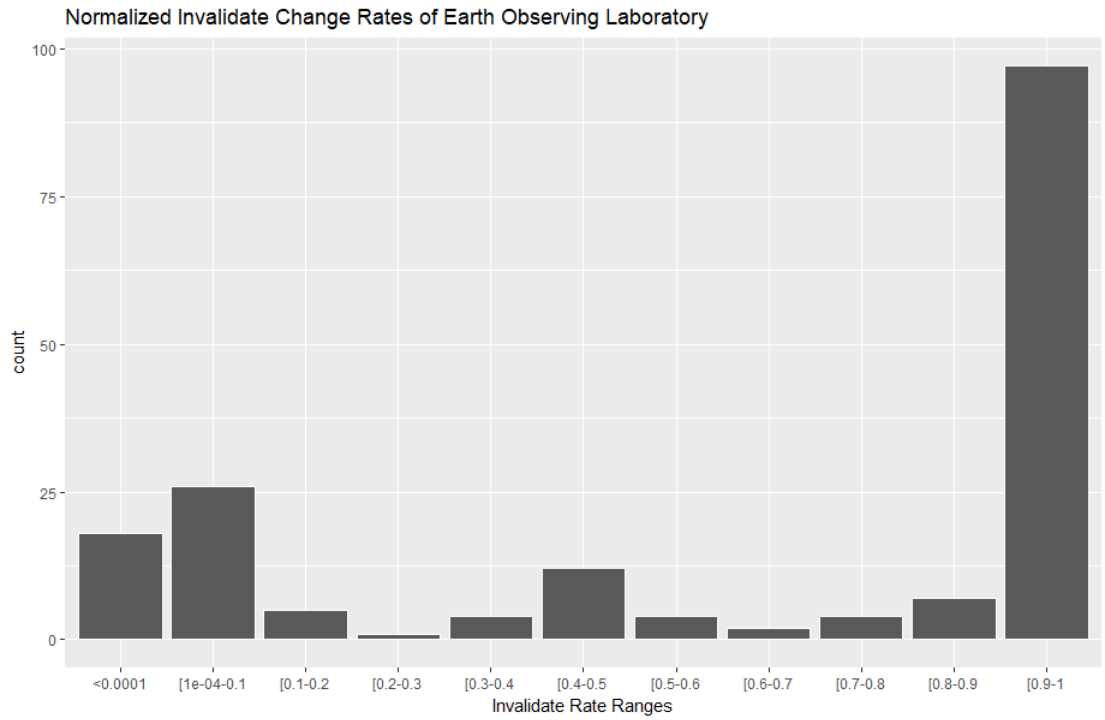


Figure 5.4: Distribution of average normalized Invalidate counts for each data set in Eath Observing Laboratory.

a quarter of the data sets added close to the original data sets' size of files suggests that the entire data set is being replaced. **Invalidations** do not have outliers since only files within the data set can be removed. The data is extremely biased with only 0.04 separating the median and maximum value. From Table 5.3, at least a quarter of values are 1. Figure 5.4 also shows a muted distrubtion around 0.5.

Figure 5.5, representing the normalized **Modify** distribution, is almost a mirror of the **Invalidation** chart. The right bar is specifically cut off to capture only 0s, showing that almost a majority of data sets modify 0 files, having 0 files that share names between versions. The distribution is consistent with a practice of removing all the files in a data set and replacing the files with a new data set using different filenames. The second feature of this graph shows around 40 data sets in which all or almost all files match across versions. A small spike of data sets are centralized around 0.5, very much like the other normalized change graphs.

The high concentration of data sets towards 1 in **additions** and **invalidations** suggests a more complicated interaction within the data sets. Individually,

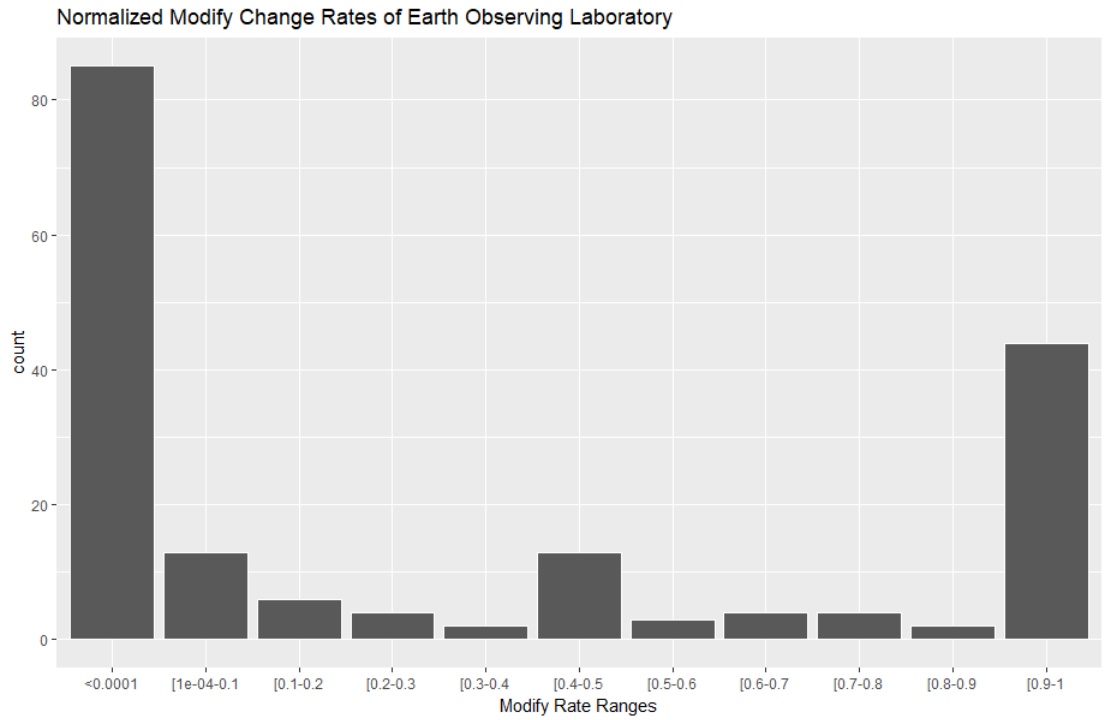


Figure 5.5: Distribution of average normalized Modify counts of each data set in Eath Observing Laboratory.

the normalized distributions do not show the connection between all three changes since the changes share a common feature, the version transition the changes describe. Together, the AIM changes create a coordinate in three dimensional space, showing the inter-relation of the changes. Figure 5.6 shows a scatter plot grouping unnormalized change counts for each version. Unlike the other charts, the size of the changes are not normalized by data set size, but the values have the log1p function applied to account for a heavy bias towards 12 and 13. Notice the one-to-one trend between **Adds** and **Invalidates** which shows the tendency of data sets to replace every file and assign a new filename. If the two changes did not co-occur, a normalized **Add** score of 1 would indicate that data sets tend to double in size instead. The files are more likely to retain filenames when only a few files in a data set are being modified.

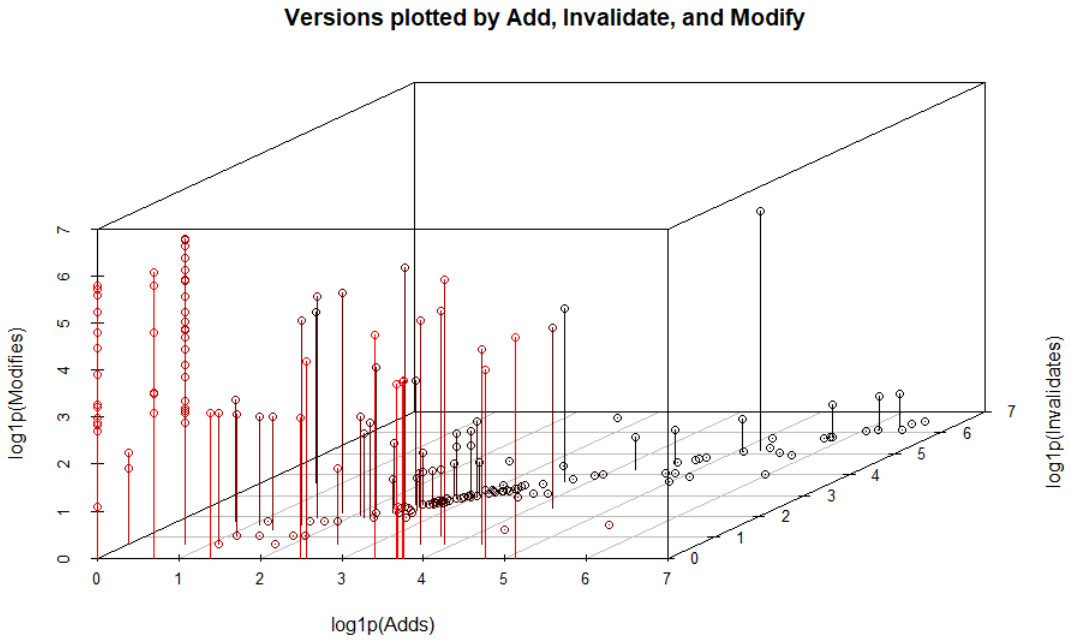


Figure 5.6: Distribution of average normalized Modify counts of each data set in Eath Observing Laboratory.

5.5 Analysis

5.5.1 Impact Assessment Change Counts

The impact assessments obtained for GCMD Version 8.2, 8.3, and 8.4 revealed that some of the changes began much earlier than the valid duration of the previous version. Adjusting for the new duration, the change rates reduce to under 0.5, aligning with the values in Cluster 2. The impact assessments furthermore provide change counts in the format of **Add**, **Invalidate**, and **Modify**. In none of the cases, shown in Table 5.4, did the metrics completely align although Version 8.3 came close with a difference of 3. VersOn does not consistently overestimate or underestimate across the versions, but the assessments most consistently align on **Invalidations** which make up very few of the changes.

A investigation into the specific differences in Version 8.3 revealed that the term “Saline Lake” does not appear in the change log, but “Leaf Area Index (LAI)” appears twice. LAI appears twice because it has two unique identifiers. Six terms appeared in the change log as **Modifications** but missed three terms from the

Table 5.4: Differences in VersOn and Impact Assessment metrics

Version	Add	Invalidate	Modify
8.2(VO)	53	1	26
-8.2(IA)	48	0	4
	5	**1**	**22**
8.3(VO)	58	0	13
-8.3(IA)	58	0	10
	0	**0**	**3**
8.4(VO)	53	0	1
-8.4(IA)	66	0	5
	-13	**0**	**-4**
8.5(VO)	68	2	22
-8.5(IA)	55	0	30
	13	**2**	**-8**

impact assessment. The primary driver between the differences lies in impact assessments being sourced from community requests. The focus of the change analysis becomes arranged around the preferred label rather than the unique identifier used to implement the keyword. Impact assessments capture changes that modify a keyword's label and that doesn't change the keyword's place in the taxonomy as a result. The change log uses a keyword's unique identifier and its placement in the taxonomy to determine changes to the structure. The difference in metrics collection once again illustrates the producer/consumer dynamic in data version management as well as the need for clear versioning practices. While the comparison between the two counts would be invalid due to differences in practice, a valid comparison could be constructed using the unique identifier to align entries and just the preferred label to determine if the keywords differ.

5.5.2 Hidden Volatility

To determine if the actual change rate is being obscured by the version publication rate, the duration of each version must be calculated. Since only one version is published at the end of that duration, simply taking the inverse of the duration will give the version publication rate. The change counts are then multiplied by the version publication rate for each version to acquire the change rate for each version. Because the change counts are often the size of the data sets, as shown in Figure

Table 5.5: Summary of Kolmogorov-Smirnov Test results for Earth Observing Laboratory.

	Add	Invalidate	Modify	Versions
Length	205	192	114	227
D-Value	0.12919	0.14464	0.19727	NA
p-Value	0.05487	0.02575	0.005443	NA

5.6, the means must be adjusted to align with the version publication rate mean in order to perform the Kolmogorov-Smirnov test. The test will determine the likelihood that the change distribution comes from the same distribution that produced the version publication rates.

Each version of a data set stored in EOL is assigned three different times, "version publish time," "version creation time," and "version modification time." Version publish time indicates the time the version was made available to the public, usually the data set was added to the database. Version creation time denotes the moment at which a version designation was given to the collection of files, beginning in mid-2014 when the versioning system was implemented. Version modification time indicates the time at which the version metadata was changed. Using version publish time most closely resembles the duration of version validity, and the following computations use version publish time.

The duration is calculated by taking the publish time of the next version and subtracting the publish time of the current version. Some of the durations needed to be filtered out to provide valid results. Due to a few coding errors in time assignments, 7 versions had to be removed because the durations were negative. Duration is measured in days, and the rate of version publication is determined by taking the inverse of the duration, giving versions per day. To acquire the AIM change rates, the changes are divided by the associated duration for each version, returning change per day. Since the change rates are biased towards at 0, the log of the rates are taken to give the values a more normal distribution. Values where an AIM change is 0 had to be removed in order to properly apply the log function. The size of each distribution can be found in Table 5.5.

Since the durations are log-normally distributed, concentrated close to 0, the log of the durations are taken to normalize the data. The log function is also applied

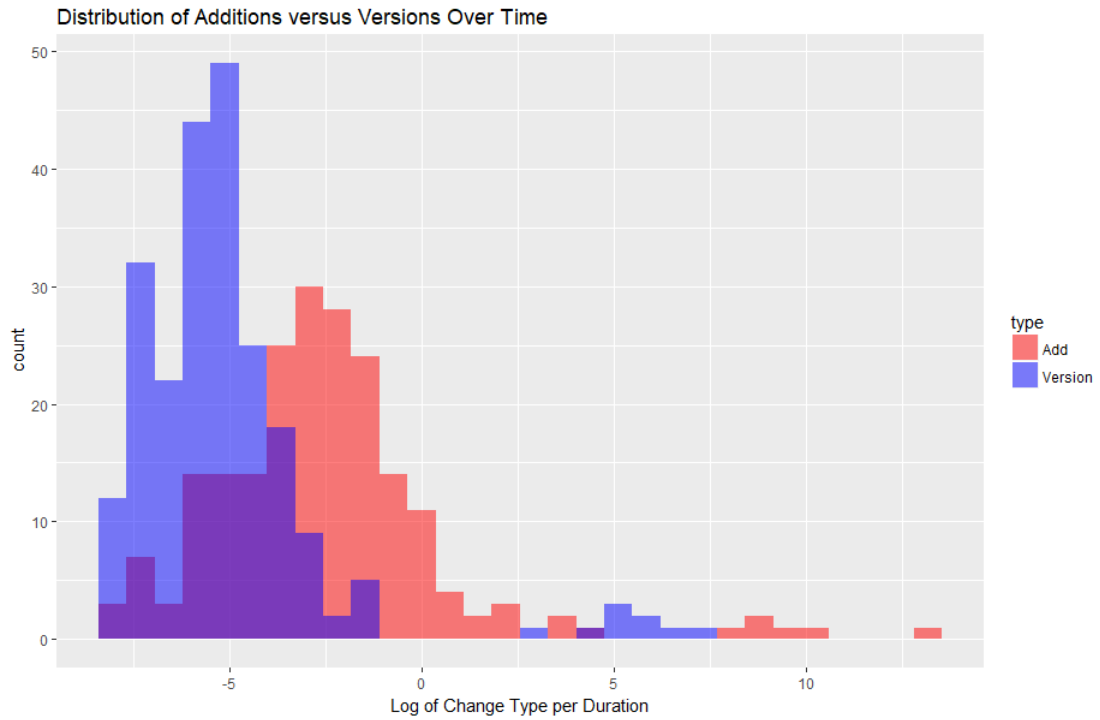


Figure 5.7: Distribution of average normalized Modify counts of each data set in Eath Observing Laboratory.

to the AIM changes after the division by the duration. The inverse of the log of the duration is taken to acquire the rate of version release. From Figure 5.7 and Figure 5.8, the change distributions, in red and green respectively, are translated a few points to the right slightly. Because the means do not not coincide well enough, the distributions must be re-aligned at the version publication rate's mean value. As noted in Section 5.4, almost half of the versions have 0 **Modifications**, meaning that the values must be filtered out. The resulting graph for comparison in Figure 5.9 shows a more flattened **Modify** distribution.

The Kolomogorov-Smirnov Test was used to determine if the **Adds**, **Invalidates**, or **Modifies** follow a distribution the same as the version publication distribution as the null hypothesis. Table 5.5 shows that the distribution of **Adds** is statistically significant with 90% confidence while **Invalidates** and **Modifies** can reject the null hypothesis with at least 95% confidence. The analysis demonstrates that there is strong evidence version publications do not accurately reflect the actual change rate of the EOL data sets.

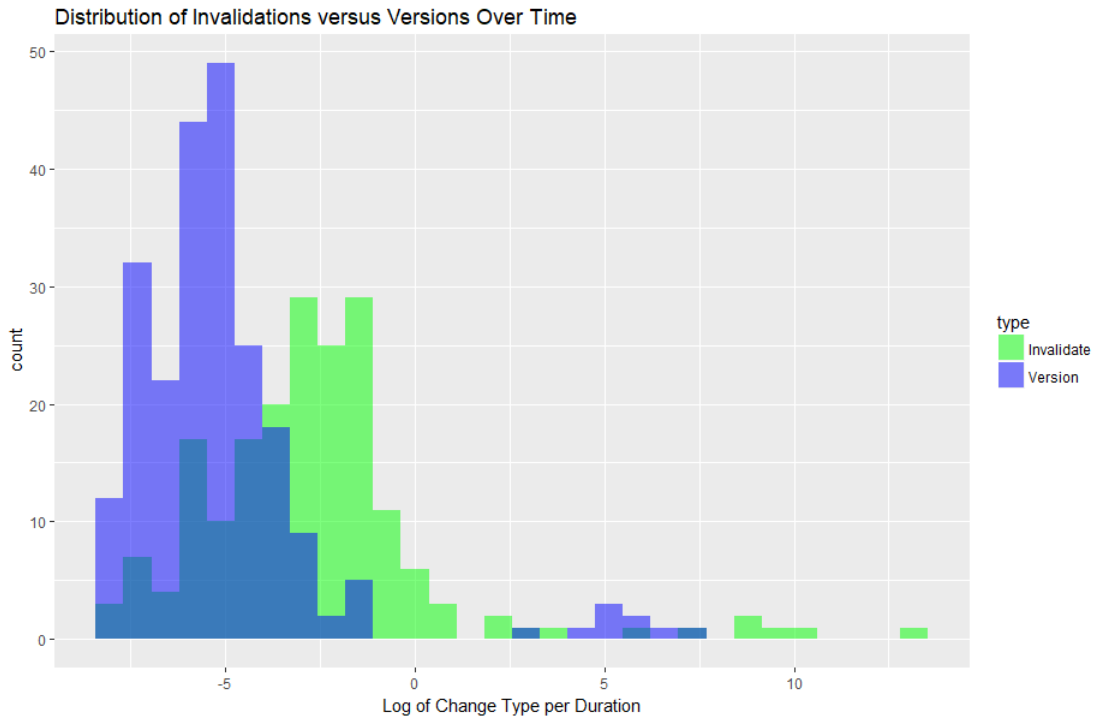


Figure 5.8: Distribution of average normalized Modify counts of each data set in Eath Observing Laboratory.

5.6 Summary

In Chapter 5, we explored different ways in which versions hide the actual change behavior within a data system. The GCMD keywords showed one perspective when evenly distributed, but spread across time, the versions have very different behavior. The change rates, once re-coupled to time, show that versions can provide a misleading view of how changes apply to a data set. Versions package together changes that can originate from times prior to the previous version, disrupting the assumed relationship between consecutive versions. GCMD also breaks assumptions when the impact assessments use different metrics to determine a data set change, reinforcing the concept that perspective and context play a major role in versioning methodology. Using only version names, Earth Observing Laboratory data is distributed uniformly by single versions, but 5.6 shows a more vibrant behavior in the data sets. The chart revealed trends in file naming and replacement. The data sets also demonstrated that AIM changes behave significantly differently than the behavior versions reveal.

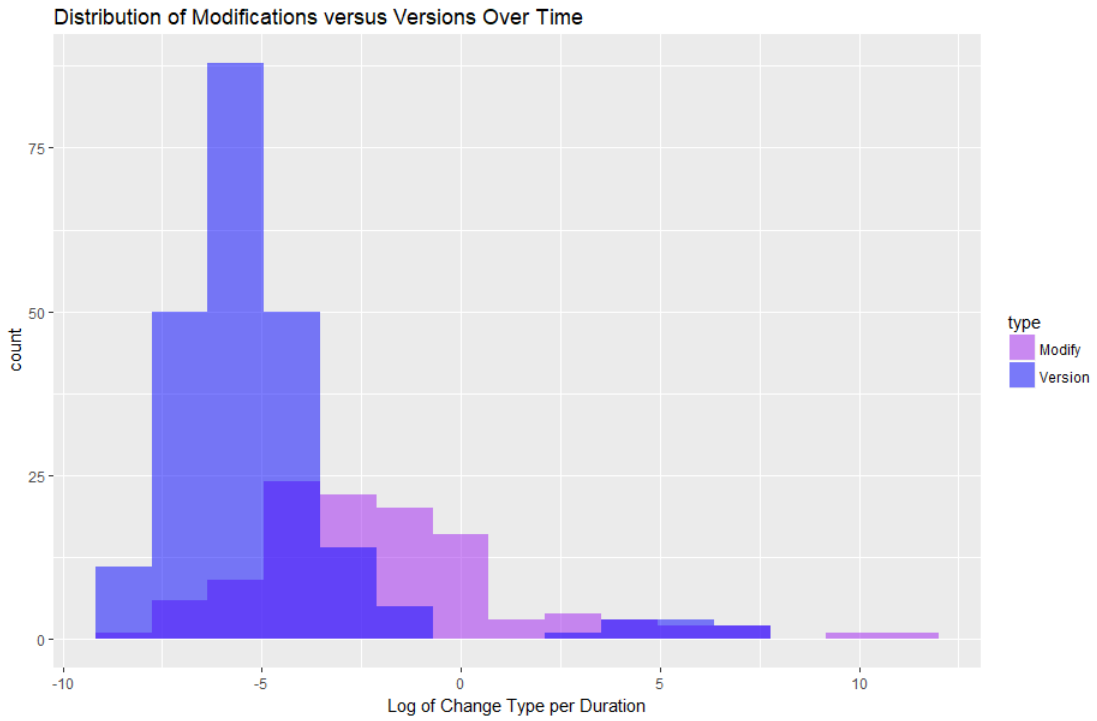


Figure 5.9: Distribution of average normalized Modify counts of each data set in Eath Observing Laboratory.

CHAPTER 6
ANALYSIS

6.1 Introduction

Implementing the versioning model yielded results more complicated than the simple model expected. While the model addresses difficulties in other linked data approaches, it requires many more triples to express the relationship. The scalability created space issues with encoded change logs, especially in JSON-LD. RDFa also proved to be a more restrictive structured data method than expected. The implementation required multiple attributes per **modification** to accommodate both row and column **attributes** associating with a cell. There were discrepancies between GCMD Keyword version identifiers and the change detected within the data set. Finally, the versioning model was used not to document sequential versions but to compare the results of different species classifiers.

6.2 Model

The versioning model's development began with an expectation that versions would be sequential. The Marine Biodiversity Virtual Laboratory (MBVL) data set demonstrated a case where four data sets were not related by temporal sequence. One is not a transformation of another since we are studying the effects of changing the taxonomy or algorithm. Additionally, since we do not know which version is the best, we cannot consider any data set as an update of the others. Finally, no entity preexisted as the data sets resulted from an ongoing analysis and further steps have not been developed. As a result, the current definition of *prov:wasDerivedFrom* would not be able to capture the relationship between these data sets. The model improves upon expressing versions in linked data by focusing on the differences between objects rather than the sequence. The model takes inspiration from *schema:UpdateAction* by dividing up the **changes** into three forms, but improves upon it by adopting the provenance model's transition from one object to the next. The resulting forms diverge from Schema.org's context of an agent acting

upon an object.

The reason *prov:Generation* and *prov:Invalidation* are not used is because they expect an activity to act upon an object. It is not generally true that an action actively adds or removes an object's attribute from in the left-hand version to produce the right-hand revision. That assumption minimizes the ability to conduct versioning comparisons between objects that are not sequentially adjacent. The PROV concepts also have a property pointing towards the responsible activity which is assumed to be the immediately preceding activity. The assumption fails to consider the case where a change propagates further changes downstream, generating or invalidating the current object. The versioning model avoids confusion by only considering the versions and their differences.

6.3 Implementation

6.3.1 Scalability

The versioning model breaks up a revision into constituent changes, acting upon different attributes of the version. Other ontologies use a single property to relate versions. While it is more specific, the VersOn implementation encounters scalable space consumption problems. PROV only requires 3 to 5 triples in order to make a *prov:wasRevisionOf* statement. This model uses 9 triples for a *vo:ModifyChange* and 7 to encode *vo:AddChange* and *vo:InvalidateChange*. An implementation of the model, therefore, has space complexity of $O(7M + 5(A + I))$ since declaring version objects takes a constant two statements. 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 *schema:AddAction*. Thus the primary benefit of using this model comes from semantics.

6.3.2 Structured Data and the Model

While machine-readable change logs have always been a desired goal of this dissertation work, their requirements diverged from the versioning model's needs. The model, as a result, leverages very little from visible content on the change log. Symmetric representation in the log also made encoding the graph using RDFa challenging without explicitly defining the whole graph in invisible span tags. Adherence with a log oriented approach would also likely have reduced the number of statements needed to form the versioning graph. The resulting ontology would likely be a collection of properties and concepts to use in annotating a document.

The current model construction provides great flexibility for version and distance capture. The model adapts to multiple attributes smoothly. Greater adherence to structured data adoption may need to come in the form of graph simplification or metered release of new editions to ensure that change logs do not grow too large.

6.4 Distance Measure

As mentioned in Section 1.6, a version model provides the framework, provenance models provide the context, and change logs fill in the gap between versions. Change logs, therefore, provide the most substance to quantify the distance between versions. The automated log generation additionally ensures this by including all differences into the change log. Anything unmapped remains the same between versions and does not contribute towards the distance. While MBVL demonstrated a case where domain knowledge could be added to the versioning graph and provide context for distances, other applications may not demonstrate the same amount of uniformity within changes. More domain information and reasoning may be necessary to determine if one add change significantly more impactful than others in a versioning graph.

6.5 Summary

The versioning model uses expanded semantics to better capture the differences between versions. When implemented in JSON-LD, the versioning graph

integrates well with text change logs, but it must address scalability issues with more volatile data sets. The model's construction allows multiple versions to be linked together into a single graph, but graphs with four or more versions may have problems with discontinuous attributes. The implementation was not able to provide evidence linking change counts to version identifiers due to strong disagreement with GCMD Keywords version 8.5. The results do indicate that version identifiers need better quantitative support. The MBVL results also demonstrate that the versioning model can provide comparisons in more contexts than documentation.

CHAPTER 7
Discussion & Conclusion

7.1 Hidden Versioning Cost

The evidence that generation costs hamstring the availability of change logs with data sets by default grows with the findings in Section 3.6. In Table 3.3, even the plain text change log is twice the size of the data sets the document describes. The trade-off of sufficient documentation versus available resources comes into play. In the end, work to bridge versions with the Noble Gas data set is offloaded onto the data consumer when transitioning between versions. Mitigating the storage and consumption costs becomes the highest priority in getting widespread change log adoption.

The tradeoff becomes especially important because change documentation often does not have immediate value. Until consumers have a need to refer back to historical data, the documentation goes unused. The availability of the information allows consumers to trace changes at crucial times, becoming indispensable in the future. Automated change log production also produces more complete documentation that may not be fully captured by human error. The ability to address and predict the costs of implementing a versioning system requires a significantly better understanding of data set version behavior and fundamentals.

7.2 Producer/Consumer Versioning Dynamic

The investigation into GCMD Keywords has demonstrated the importance of investigating beyond sequential version releases. The initial hypothesis was that the dominant change count could provide a reliable indicator to differentiate major and minor versions. The resulting numbers shows some reflection of the version name in the change counts. A more important finding shows that different approaches can be used to evaluate the number of changes in the Version 8.4 to 8.5 transition. The difference highlights a barrier between expertise of data producers and consumers within a system. Without prior knowledge of the namespace change, the version

indicator violates the GCMD Keyword data policy. The ability for a consumer to determine the amount of change within a system becomes incredibly important as the associated change document dictates to the data consumer how the producer thinks users should interact with the data.

The GCMD Keyword data set also demonstrates a transparency issue when utilizing a sequential versioning scheme since versions are not bound to a temporal or change count schedule. In Figure 4.5, we can see that there is a sharp drop off in change counts once entering variants of the 8th major release. The finding that the change counts do not consistently relate with version identifiers has already been discussed, but the chart is misleading in showing each version equally spaced from the others. Temporally, the versions are separated by a variety of durations. As mentioned in Section 2.4, the release rate of versions can be artificially controlled, disconnecting the rate of change from time. When refactoring time back into the change measurement, we can see very distinct separation in the change rate as well as their conformance with the version identifiers assigned at the end of the change period. In particular, we can see in Cluster 2 of Figure 5.2 that versions can be arbitrarily released in quick succession even though work on the changes inside the version began in 2008, 2014, or 2015. This finding indicates that version releases cannot be universally trusted to provide a complete picture of the change within a system by itself.

While investigating inconsistencies between change counts found by the change log and those reported by the impact assessment, differences between the metrics became apparent. The lack of alignment arose from a difference between the way the community sees and proposes the keywords and the way the keywords are digitally encapsulated and stored in the KMS. As a result, the impact assessments do not capture the structural changes that result from additions to the taxonomy.

7.3 Hiddent Data Volatility

The EOLs small data set size allows it to adopt a comprehensive replacement method. The versioning model identified the need for unique file identifiers to determine when files are specifically changed which were not part of the original

versioning metadata starting in 2014. The process of capturing change within the system using the model naturally led to a set of basic requirements necessary to implement a versioning system.

The 3 dimensional scatter plot in Figure 5.6 shows a very surprising tendency in EOL data sets. While the description of update methods suggests data sets should be modify dominant, many of the versions replace and rename all the files in a version. The volatility analysis for these versions show that when a version is made it will likely entirely replace the previous version. The trend also suggests a concerning behavior of contributing scientists to transition away from a previously established file naming scheme.

The problem with change hiding is that version releases mask a data sets true volatility. From the Kolomogorov-Smirnov test results, each of the change types demonstrated a different distribution from the visible version release rate.

7.4 New Versioning Nomenclature

Analysis of versioned data sets has revealed three types of data, dependent on the way in which versions are released: single, periodic, and intermittent. Single version data sets contain data which cannot be replicated or in which modification would entirely invalidate the data. High energy physics, previously mentioned, and surveillance data fit within this category. The data sets in this category will usually only experience additions and invalidations since scientists cannot change the data.

Periodic data sets exhibit version releases at regular intervals in time. Large data collections usually exhibit a regular behavior when they follow a periodic data collection scheme. The ARM data center releases data at daily intervals, meaning new versions every day. The reasons that ARM data sets are not overloaded with version numbers is that some operations, in this case new files, are masked to increase the pertinence of each version designation. The problem that masking additions causes is the actual amount of change within the data set over time also becomes masked. The data set then appears to be intermittent when it actually undergoes periodic changes. As seen in GCMD Keywords and EOL, changes are not necessarily evenly distributed among versions. The changes, as a result, are also not evenly

distributed across time. As mentioned with distributed versioning methods, periodic version releases can be used to control the volatility of a data set by collecting many changes over time before publication. Periodic data sets expend version identifiers very quickly since they must release a version even if few significant changes have occurred.

The final type of data set follows intermittent versioning which is characterized by releasing versions as appropriate or as necessary. The data sets are not bound by an established release schedules. In the intermittent category falls GCMD Keywords, the Copper data set, and the Noble Gas data set. Irregular version releases allows data managers the freedom to reduce the number of versions necessary to manage the data set. When data managers wait too long to release a new version, the number of changes in a single transition can overwhelm methods to track modifications to the data as seen in the Noble Gas data set. Since intermittent versions are not released based on time, it is very important that versions are released based on some other quantitative measure of change. Failing to do so invites unclear or worse arbitrary distinction between versions. GCMD Keywords define clear requirements for major and minor version releases, but the governance document does not explain the requirements for sub-minor versions which occasionally appear in the keyword repository.

Each data set type can additionally be sub-divided into two categories based on the observations made with the AIM model: Add dominant and Modify dominant. In the data sets currently studied, none exhibit behavior suggesting an Invalidate dominant data set. A data set is either Add or Modify dominant when a majority of versions have a majority of either Adds or Modifies. Add dominance indicates that the data is primarily growing while Modify dominance shows that a data sets coverage is primarily stable but occasionally undergoes adjustments. The GCMD Keywords is an example of an Add dominant data set since all its version transitions are comprised of new concepts. The Noble Gas data set shows modify dominance.

7.5 Conclusion

Change analysis for data sets need a great amount of work as big data sets become more common. Terms and practices need to be standardize and formalized which begins with producing discoverable and consumable change documentation. The procedures explored showed promise using linked data models, but suffered from size bloating necessary to make the documents machine consumable. Once computable, producers can begin providing better quantitative measure for change in data, but analysis has shown that the perceived change may differ depending on the consumer of the data set. The experience highlights an obscured dynamic in change information between data producers and data consumer in which producers often dictate the means of evaluating version change. Diverging from version-primary practices and including more detailed change accounting becomes a priority after discovering that versions can hide trends in the actual change rate. The difference between data set change rate and behavior suggests that future research is necessary to determine if the differences indicate versioning practices also need to be different.

CHAPTER 8
FUTURE WORK

A number of concerns were not addressed during the versioning graph research process. Since a new change statement is made for each difference between versions, some optimizations must be made to keep version graphs small enough to be encoded within change logs. Discontinuous attributes across multi-version graphs creates a problematic barrier to graph queries. Finally, further study must be done to determine methods in providing quantitative basis for version identifiers. These un-addressed questions form the most immediately approachable next steps for this versioning graph approach.

8.1 Change Log Optimization

Very large change logs encoded with JSON-LD through HTML began experiencing performance issues due to the extreme number of modifications in the graph. One observation is that a modification in one cell of the Noble Gas data set sometimes also occurs in every other cell in that spreadsheet column. The relation of all those cells could then be summarized with a singe modification statement with just the column attribute, reducing the space utilization dependency from the number of rows to a single statement. The summarization could reduce the change log's size to a manageable enough level to be viewable.

8.1.1 Dynamic Change Logs

Users selectively use portions of particularly expansive data sets to filter data down to their region of study. Tools can use the versioning model to identify pertinent sections of a large change log and parse out the extraneous entries. Means to isolate change activities are necessary for users to determine the impact a new version has on the operation of their workflow. The versioning graph can also contribute to the generation of unique change logs to accompany dynamically created data sets. As mentioned in Section 2.2.4, users can dynamically aggregate and filter

data sets to produce a new unique set of data, but doing so still requires tracking of differences from the original data set or sets. Further work will need to be done determining requirements to automate change log creation for these data sets.

8.2 References to Bug Tickets

As mentioned previously, data versioning plays a major role in documentation for bug fixes and audits. Similar to the work done linking Bugzilla and GIT, bug tracking and change logs should also be connected. The linked data approach taken to develop versioning graphs provides an avenue to link the data of versions and bug trackers together. The work would add value to users' research by laying out which changes address bugs consumers have reported or found. The practice would also reinforce producer culpability and responsiveness to the user community. Linking data change and bug information would also allow data producers to document the evolution of their data in response to error corrections. The traceability would let producers determine if a bug is new or recurring as well.

8.3 Supervised Versioning

The comparisons utilized in executing change log creation were basic string or numerical comparisons. More complex data sets may need supervised input to properly model version changes. Attributes that split or merge were not tested or evaluated for whether they needed unique behavior to capture.

8.4 Multi-version Graphs

At present, the versioning model captures only changing as a matter of convention and to save space. Version graphs with multiple versions can suffer discontinuities across attributes which don't change between two versions, but then experience a modification later. Discontinuities in the graph causes problems for search queries since a directed path does not exist through all versions in the graph for that attribute. The definition of a null-step to bridge gaps could provide a temporary solution to show an attribute in the graph hasn't changed but re-establish connectivity. The addition could also introduce new space utilization concerns.

Once standardized, multi-version graphs provide a full history of a work. Versioning systems often only need to provide the changes between two specific versions. Not all changes along that profile is necessary. As a result, reasoning methods need to be developed to help summarize changes across multiple versions.

8.5 Change Distance and Dot-decimal Identifiers

The initial research to study the relationship between change counts and version identifiers broke down due to the subjectivity of identifier assignment. Not enough evidence was found to determine if identifiers were assigned accurately. Applying the versioning model to more data sets and comparing change counts may be necessary to determine what quantifiable methods, if any, can be used as a basis for version identifier assignment. The research would be conducted to determine the extent to which dot-decimal identifiers can communicate change of a data set.

Compressing changes into a single row or column may disproportionately affect resulting distance counts.

8.6 Other Methods of Change Distance Calculation

Flow calculation seems possible.

8.7 Database Context

One area not explored by the work in this dissertation is the context of centralized databases. While they resemble spreadsheets, centralized databases only have a single instance and use multiple tables which are routinely merged to answer queries. The scripts and process used to version spreadsheets would not work on these databases since the data is not instanced. The databases, however, use standardized add, delete, and modify commands which do map to the versioning model. Work remains to be done in studying how these commands can be captured and output as a versioning graph instead of using the script to perform the comparison.

8.8 Implementing Recursive Tiers

The multi-tiered nature of versioning models have been mentioned multiple times, but the specified versioning model only defines one tier. Multiple tiers may be necessary to capture the granularity of some data sets such as the one illustrated by Barkstrom. If attributes are also allowed to be versions, graphs can be nested recursively to form a multi-tiered graph. More work needs to be done to understand what such a graph would look like as well as the mechanics necessary to make the graph accessible.

8.9 Multi-file Versions

As pointed out in Chapter 4, little guidance is given on versions spread across multiple files. For the Noble Gas data set, the files were grouped into a collection, but if the desired representation are separate objects and multiple left-hand versions, not much work has been done to explain how that should be implemented. An alternative way to implement the Noble Gas graph would be to have the collection link not to the attributes and changes, but to a file which then behaves like a left-hand version in the normal graph. Such a construction is possible, but whether the setup is desirable remains to be studied.

8.10 Summary

Future work should be conducted to reduce the size of change logs, re-connect multi-version graphs, and determine a quantitative basis for version identifiers. Change logs can be shortened by discovering modifications occurring over an entire column which can be summarized in a single statement. Null-step links could be used to reconnect attributes in multi-version graphs, but this may also introduce new space consumption issues. The versioning model should be applied to more data sets employing the dot-decimal identifier method to gather evidence on the extent to which the identifiers can communicate change in a data set. These approaches were left unexplored by the project's conclusion.

REFERENCES

[1] 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

[2] 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

[3] 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

[4] 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.

[5] 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

[6] T. Lebo, D. McGuinness, and S. Sahoo, "PROV-o: The PROV ontology," W3C, W3C Recommendation, Apr. 2013, http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[7] I. S. G. on the Functional Requirements for Bibliographic Records, "Functional requirements for bibliographic records," International Federation of Library Associations and Institutions, Tech. Rep., 2009.

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

[9] 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

[10] 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

[11] 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

[12] 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.

[13] 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.

[14] 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.

[15] S. Miles and Y. Gil, "PROV model primer," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/.

[16] P. Groth and L. Moreau, "PROV-overview," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/.

[17] L. Moreau and P. Missier, "PROV-dm: The PROV data model," W3C, W3C Recommendation, Apr. 2013, http://www.w3.org/TR/2013/REC-prov-dm-20130430/.

[18] P. Missier, L. Moreau, and J. Cheney, "Constraints of the PROV data model," W3C, W3C Recommendation, Apr. 2013, http://www.w3.org/TR/2013/REC-prov-constraints-20130430/.

[19] T. D. Nies and S. Coppens, "PROV-dictionary: Modeling provenance for dictionary data structures," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-dictionary-20130430/.

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

[21] H. Hua, S. Zednik, and C. Tilmes, "PROV-xml: The PROV xml schema," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/.

[22] P. Groth and G. Klyne, "PROV-aq: Provenance access and query," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-aq-20130430/.

[23] L. Moreau and P. Missier, "PROV-n: The provenance notation," W3C, W3C Recommendation, Apr. 2013, http://www.w3.org/TR/2013/REC-prov-n-20130430/.

[24] J. Cheney, "Semantics of the PROV data model," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-sem-20130430/.

[25] K. Eckert and D. Garijo, "Dublin core to PROV mapping," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-dc-20130430/.

[26] T. Lebo and L. Moreau, "Linking across provenance bundles," W3C, W3C Note, Apr. 2013, http://www.w3.org/TR/2013/NOTE-prov-links-20130430/.

[27] 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

[28] 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

[29] 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

[30] 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

[31] 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

[32] (2012, Jun.) Dcmi metadata terms. DCMI Usage Board. Accessed: February 8, 2017. [Online]. Available: http://dublincore.org/documents/2012/06/14/dcmi-terms/

[33] Updateaction. Schema.org. Accessed: January 19, 2017. [Online]. Available: http://schema.org/UpdateAction

[34] Replaceaction. Schema.org. Accessed: January 19, 2017. [Online]. Available: http://schema.org/ReplaceAction

[35] Addaction. Schema.org. Accessed: January 19, 2017. [Online]. Available: http://schema.org/AddAction

[36] Deleteaction. Schema.org. Accessed: January 19, 2017. [Online]. Available: http://schema.org/DeleteAction

[37] 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/

[38] 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.

[39] 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

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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

[46] 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

[47] 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

[48] 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

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

[50] 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

[51] 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

[52] Data versioning. Australian National Data Service. Accessed: June 9, 2017. [Online]. Available: http://www.ands.org.au/working-with-data/data-management/data-versioning

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

[54] 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

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

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

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

[58] 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

[59] 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

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

[61] 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

[62] 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.

[63] 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

[64] 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.

[65] 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

[66] 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

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

[68] 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

[69] 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

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

[71] C. Hauptmann, M. Brocco, and W. Wörndl, "Scalable semantic version control for linked data management," in *2nd Workshop on Linked Data Quality (LDQ)*, ser. CEUR Workshop Proceedings, A. Rula, A. Zaveri, M. Knuth, and D. Kontokostas, Eds., no. 1376, Aachen, 2015, accessed: February 21, 2017. [Online]. Available: http://ceur-ws.org/Vol-1376

[72] A. Rula, A. Zaveri, M. Knuth, and D. Kontokostas, Eds., *Proceedings of the 2nd Workshop on Linked Data Quality (LDQ)*, ser. CEUR Workshop Proceedings, no. 1376, Aachen, 2015. [Online]. Available: http://ceur-ws.org/Vol-1376/

[73] 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

[74] 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

[75] "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

[76] S. McCarron, I. Herman, B. Adida, and M. Birbeck, "RDFa core 1.1 - third edition," W3C, W3C Recommendation, Mar. 2015,
http://www.w3.org/TR/2015/REC-rdfa-core-20150317/.

[77] M. Sporny, I. Herman, B. Adida, and M. Birbeck, "RDFa 1.1 primer - third edition," W3C, W3C Note, Mar. 2015,
http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/.

[78] 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

[79] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindstrom. (2017, Dec.) Json-ld 1.1. W3C. Accessed: June 7, 2017. [Online]. Available:
https://json-ld.org/spec/latest/json-ld/

[80] 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

[81] 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

[82] 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

[83] 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.

[84] 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.

[85] 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.

[86] 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

[87] 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

[88] 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

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

[90] 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

[91] 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.

[92] K. Elder, D. Cline, G. E. Liston, and R. Armstrong, "Nasa cold land processes experiment (clpx 2002/03): Field measurements of snowpack properties and soil moisture," *Journal of Hydrometeorology*, vol. 10, no. 1, pp. 320–329, 2009. [Online]. Available: https://doi.org/10.1175/2008JHM877.1

[93] M. A. Parsons, M. J. Brodzik, and N. J. Rutter, "Data management for the cold land processes experiment: improving hydrological science," *Hydrological Processes*, vol. 18, no. 18, pp. 3637–3653, 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.5801

[94] 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

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

[96] A. Miles and S. Bechhofer, "SKOS simple knowledge organization system reference," W3C, W3C Recommendation, Aug. 2009, http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[97] T. Stevens, "Nasa gcmd keyword 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

[98] Marine biodiversity virtual laboratory. Accessed: September 28, 2016. [Online]. Available: https://tw.rpi.edu/web/project/MBVL

[99] Global Change Master Directory (GCMD), *Global Change Master Directory (GCMD) Keyword Governance and Community Guide Document*, version 1.0 ed., National Aeronautics and Space Administration (NASA), Aug. 2016, accessed: June 10, 2018. [Online]. Available: https://cdn.earthdata.nasa.gov/conduit/upload/5182/KeywordsCommunityGuide_Baseline_v1_SIGNED_FINAL.pdf

[100] UCAR/NCAR - Earth Observing Laboratory. (2018) About eol. UCAR/NCAR - Earth Observing Laboratory. Accessed: June 10, 2018. [Online]. Available: https://www.eol.ucar.edu/about-eol

APPENDIX A
NOBLE GAS CHANGE LOG GENERATOR VERSION 1 TO 2

```
from os.path import join, dirname, abspath, isfile
from os import sep as separator
import xlrd, sys, json
import glob
import re


def index_convert(index1):
if index1 < 17:
return index1
elif index1 < 24:
return index1+1
elif index1 < 26:
return 26+(3*(index1-24))
elif index1 < 28:
return 44+(3*(index1-26))
elif index1 < 32:
return 53+(9*(index1-28))
elif index1 < 36:
return 88+(2*(index1-32))
elif index1 < 38:
return 95+(4*(index1-36))
elif index1 < 41:
return 102+(2*(index1-38))
elif index1 < 43:
return 112+(2*(index1-41))
elif index1 == 43:
return 172
elif index1 < 50:
return 174+(3*(index1-44))
```

```
elif index1 < 54:
return 191+(index1-50)
else:
print 'Error: Out of bounds'
return -1

def test_alignment():
for i in range(0, 54):
print 'version2: {:5} version1: {:5}'.format(i, index_convert(i))

def compare_print(mode, key, val1, val2, v1_file, v1_index = 0,
    v2_index = 0, changelog = None):
if changelog:
if mode == 'r':
out = u'''        <tr  about="Change{}{}" typeof="vo:ModifyChange">
<td align="right" rev="vo:Undergoes" resource="v1:Attribute{}{}v1"
    typeof="vo:Attribute">{:2}({})</td>
<td property="vo:resultsIn" resource="v2:Attribute{}{}v2"
    typeof="vo:Attribute">{:2}</td>
<td>{:>10}</td>
<td>{:>10}</td>
<span about="Version1" property="vo:hasAttribute"
    resource="v1:Attribute{}{}v1"></span>
<span about="Version2" property="vo:hasAttribute"
    resource="v2:Attribute{}{}v2"></span>
</tr>\n'''.format(key, v2_index, key, v1_index, v1_index, v1_file,
    key, v2_index, v2_index, val1, val2, key, v1_index, key,
    v2_index)
elif mode == 'j':
out = u'''        <tr  id="ModifyChange{}{}">
<td align="right">{:2}({})</td>
<td>{:2}</td>
<td>{:>10}</td>
<td>{:>10}</td>
<script type="application/ld+json">\n'''.format(key, v2_index,
    v1_index, v1_file, v2_index, val1, val2)
```

```
59 elif mode == 't':
60 out = u"{:2}({})\t{:2}\t{:>10}\t{:>10}\n".format(v1_index, v1_file,
    ↪ v2_index, val1, val2)
61 elif mode == 'u':
62 out = u"""<http://example.com/NG/Version1> vo:hasAttribute
    ↪ <http://example.com/NG/Version1/%s> ;
63 vo:hasAttribute <http://example.com/NG/Version1/Column%i> .
64 <http://example.com/NG/Version1/%s> a vo:Attribute ;
65 vo:undergoes <http://example.com/Changelog#ModifyChange%s%i> .
66 <http://example.com/NG/Version1/Column%i> a vo:Attribute ;
67 vo:undergoes <http://example.com/Changelog#ModifyChange%s%i> .
68 <http://example.com/Changelog#ModifyChange%s%i> a vo:ModifyChange ;
69 vo:resultsIn <http://example.com/NG/Version2/%s> ;
70 vo:resultsIn <http://example.com/NG/Version2/Column%i> .
71 <http://example.com/NG/Version2> vo:hasAttribute
    ↪ <http://example.com/NG/Version2/%s> ;
72 vo:hasAttribute <http://example.com/NG/Version2/Column%i> .
73
74 """%(key, v1_index, key, key, v2_index, v1_index, key, v2_index, key,
    ↪ v2_index, key, v2_index, key, v2_index)
75 changelog.write(out.encode('utf8'))
76 if mode == 'j':
77 json1 = {
78 "@context":context,
79 "@type":"vo:Attribute" ,
80 "@id":"".join(["http://ngdb.com/v1/Attribute", key, str(v1_index)]) ,
81 "label":key ,
82 "undergoes":"".join([host, "ModifyChange", key, str(v2_index)]) ,
83 "@reverse" :    { "hasAttribute" : "Version1" }
84 }
85 json2 = {
86 "@context":context,
87 "@type":"vo:ModifyChange",
88 "@id":"".join([host, "ModifyChange", key, str(v2_index)]) ,
89 "resultsIn":"".join(["http://ngdb.com/v2/Attribute", key,
    ↪ str(v2_index)])
```

```
90  }
91  json3 = {
92  "@context":context,
93  "@type":"vo:Attribute" ,
94  "@id":"".join(["http://ngdb.com/v2/Attribute", key, str(v2_index)]) ,
95  "label":key ,
96  "@reverse" :    { "hasAttribute" : "Version2" }
97  }
98  json.dump([json1, json2, json3], changelog, indent=4, sort_keys=True)
99  changelog.write('''
100 </script>
101 </tr>
102 ''')
103 else:
104 print '{:5}  version1: {:10} version2: {:10}'.format(key, val1, val2)
105
106 labels = {17:"SAMPLING - DEPTH - >,<",
107 25:"[He] - ppm - >,<", 27:"[He] - ppm - err", 28:"[He] - mkcc/ -
    ↪  >,<", 30:"[He] - mkcc/ - err", 31:"[He] - mol/ - >,<", 32:"[He] -
    ↪  mol/ - L H2O", 33:"[He] - mol/ - err",
108 34:"[He+Ne] - ppm - >,<", 35:"[He+Ne] - ppm", 36:"[He+Ne] - ppm -
    ↪  err", 37:"[He+Ne] - mkcc/ - >,<", 38:"[He+Ne] - mkcc/ - g H2O",
    ↪  39:"[He+Ne] - mkcc/ - err", 40:"[He+Ne] - mol/ - >,<",
    ↪  41:"[He+Ne] - mol/ - L H2O", 42:"[He+Ne] - mol/ - err",
109 43:"[Ne] - ppm - >,<", 45:"[Ne] - ppm - err", 46:"[Ne] - mkcc/ -
    ↪  >,<", 48:"[Ne] - mkcc/ - err", 49:"[Ne] - mol/ - >,<", 50:"[Ne] -
    ↪  mol/ - L H2O", 51:"[Ne] - mol/ - err",
110 52:"[20Ne] - ppm - >,<", 54:"[20Ne] - ppm - err", 55:"[20Ne] - mkcc/
    ↪  - >,<", 56:"[20Ne] - mkcc/ - g H2O", 57:"[20Ne] - mkcc/ - err",
    ↪  58:"[20Ne] - mol/ - >,<", 59:"[20Ne] - mol/ - L H2O", 60:"[20Ne]
    ↪  - mol/ - err",
111 61:"[Ar] - ppm - >,<", 63:"[Ar] - ppm - err", 64:"[Ar] - mkcc/ -
    ↪  >,<", 65:"[Ar] - mkcc/ - g H2O", 66:"[Ar] - mkcc/ - err",
    ↪  67:"[Ar] - mol/ - >,<", 68:"[Ar] - mol/ - L H2O", 69:"[Ar] - mol/
    ↪  err",
```

```
112 70:"[Kr] - ppm - >,<", 72:"[Kr] - ppm - err", 73:"[Kr] - mkcc/ -
    ↪ >,<", 74:"[Kr] - mkcc/ - g H2O", 75:"[Kr] - mkcc/ - err",
    ↪ 76:"[Kr] - mol/ - >,<", 77:"[Kr] - mol/ - L H2O", 78:"[Kr] - mol/
    ↪ err",
113 79:"[Xe] - ppm - >,<", 81:"[Xe] - ppm - err", 82:"[Xe] - mkcc/ -
    ↪ >,<", 83:"[Xe] - mkcc/ - g H2O", 84:"[Xe] - mkcc/ - err",
    ↪ 85:"[Xe] - mol/ - >,<", 86:"[Xe] - mol/ - L H2O", 87:"[Xe] - mol/
    ↪ err",
114 89:"3He/4He - (R/Ra)me - err", 91:"3He/4He - (R/Ra)corr - err",
    ↪ 93:"3He/4He - Rme - E-8 - err", 96:"3He/4He - Rcorr - E-8 - err",
    ↪ 97:"Rank",
115 98:"He/Ne - >,<", 100:"He/Ne - >,<", 101:"4He/20Ne - >,<",
    ↪ 103:"4He/20Ne - err", 105:"20Ne/22Ne - err", 107:"21Ne/22Ne -
    ↪ (xE-2) - err", 108:"21Ne/20Ne", 109:"21Ne/20Ne - err",
116 110:"22Ne/20Ne", 111:"22Ne/20Ne - err", 113:"38Ar/36Ar - err",
    ↪ 115:"40Ar/36Ar - err", 116:"delta(40Ar)rad", 117:"delta(40Ar)rad
    ↪ - err",
117 118:"He/Ar - He/ - /Ar(air) - >,<", 119:"He/Ar - He/ - /Ar(air)",
    ↪ 120:"He/Ar - He/ - /Ar(air) - err", 121:"He/Ar - 4He/ - /36Ar -
    ↪ >,<", 122:"He/Ar - 4He/ - /36Ar", 123:"He/Ar - 4He/ - /36Ar -
    ↪ err",
118 124:"He/Ar - 4He/ - /40Ar(air) - >,<", 125:"He/Ar - 4He/ -
    ↪ /40Ar(air)", 126:"He/Ar - 4He/ - /40Ar(air) - err",
119 127:"f(He)=(He/Ar)s/(He/Ar)air - >,<",
    ↪ 128:"f(He)=(He/Ar)s/(He/Ar)air", 129:"f(He)=(He/Ar)s/(He/Ar)air -
    ↪ err",
120 130:"Ne/Ar - Ne/ - /Ar(air) - >,<", 131:"Ne/Ar - Ne/ - /Ar(air)",
    ↪ 132:"Ne/Ar - Ne/ - /Ar(air) - err", 133:"Ne/Ar - 20Ne/ - /36Ar -
    ↪ >,<", 134:"Ne/Ar - 20Ne/ - /36Ar", 135:"Ne/Ar - 20Ne/ - /36Ar -
    ↪ err",
121 136:"Ne/Ar - 20Ne/ - /40Ar(air) - >,<", 137:"Ne/Ar - 20Ne/ -
    ↪ /40Ar(air)", 138:"Ne/Ar - 20Ne/ - /40Ar(air) - err", 139:"Ne/Ar -
    ↪ 22Ne/ - /36Ar - >,<", 140:"Ne/Ar - 22Ne/ - /36Ar", 141:"Ne/Ar -
    ↪ 22Ne/ - /36Ar - err",
122 142:"Ne/Ar - 22Ne/ - /40Ar(air) - >,<", 143:"Ne/Ar - 22Ne/ -
    ↪ /40Ar(air)", 144:"Ne/Ar - 22Ne/ - /40Ar(air) - err",
```

```
123 145:"f(Ne)=(Ne/Ar)s/(Ne/Ar)air - >,<",
    ↪ 146:"f(Ne)=(Ne/Ar)s/(Ne/Ar)air", 147:"f(Ne)=(Ne/Ar)s/(Ne/Ar)air -
    ↪ err",
124 148:"Kr/Ar - Kr/ - /Ar(air) - >,<", 149:"Kr/Ar - Kr/ - /Ar(air)",
    ↪ 150:"Kr/Ar - Kr/ - /Ar(air) - err", 151:"Kr/Ar - 84Kr/ - /36Ar -
    ↪ >,<", 152:"Kr/Ar - 84Kr/ - /36Ar", 153:"Kr/Ar - 84Kr/ - /36Ar -
    ↪ err",
125 154:"Kr/Ar - 84Kr/ - /40Ar(air) - >,<", 155:"Kr/Ar - 84Kr/ -
    ↪ /40Ar(air)", 156:"Kr/Ar - 84Kr/ - /40Ar(air) - err",
126 157:"f(Kr)=(Kr/Ar)s/(Kr/Ar)air -
    ↪ >,<",158:"f(Kr)=(Kr/Ar)s/(Kr/Ar)air",
    ↪ 159:"f(Kr)=(Kr/Ar)s/(Kr/Ar)air - err",
127 160:"Xe/Ar - Xe/ - /Ar(air) - >,<", 161:"Xe/Ar - Xe/ - /Ar(air)",
    ↪ 162:"Xe/Ar - Xe/ - /Ar(air) - err", 163:"Xe/Ar - 132Xe/ - /36Ar -
    ↪ >,<", 164:"Xe/Ar - 132Xe/ - /36Ar", 165:"Xe/Ar - 132Xe/ - /36Ar -
    ↪ err",
128 166:"Xe/Ar - 132Xe/ - /40Ar(air) - >,<", 167:"Xe/Ar - 132Xe/ -
    ↪ /40Ar(air)", 168:"Xe/Ar - 132Xe/ - /40Ar(air) - err",
129 169:"f(Xe)=(Xe/Ar)s/(Xe/Ar)air - >,<",
    ↪ 170:"f(Xe)=(Xe/Ar)s/(Xe/Ar)air", 171:"f(Xe)=(Xe/Ar)s/(Xe/Ar)air -
    ↪ err",
130 173:"H2 - >,<", 175:"H2 - ppm - err", 176:"O2 - >,<", 178:"O2 - ppm -
    ↪ err", 179:"N2 - >,<", 181:"N2 - ppm - err", 182:"CO2 - >,<",
    ↪ 184:"CO2 - ppm - err", 185:"CH4 - >,<", 187:"CH4 - ppm - err",
131 188:"H2S - >,<", 190:"H2S - ppm - err"}
132
133 context = "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld"
134 host =
    ↪ "http://orion.tw.rpi.edu/~blee/provdist/NobleGas/changelog_json.html#"
135 #test_alignment()
136
137
138 #print v2_row[0].value
139 #print indicator_map[v2_row[0].value]
140
141 #v1_workbook = xlrd.open_workbook(v1_file)
```

```
142 #v1_sheet = v1_workbook.sheet_by_index(0)
143 #v1_row = v1_sheet.row(4)
144
145 def write_modify(r1, r2, workbook, f_out, mode):
146 if mode == 'r':
147 out = u'''  <div about="Version1" rel="vo:hasAttribute">
148 <div resource="v2:%s" typeof="vo:Attribute">
149 <span style="font-weight:bold"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">%s</span>
150 <table rel="vo:Undergoes">
151 '''%(r2[0].value, r2[0].value)
152 elif mode == 'j':
153 out = u'''
154 <div about="v2:%s">
155 <span style="font-weight:bold"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">%s</span>
156 <table>
157 '''%(r2[0].value, r2[0].value)
158 elif mode == 't':
159 out = u"%s\n"%(r2[0].value)
160 elif mode == 'u':
161 out = u""
162
163 if mode == 'r' or mode == 'j':
164 out = out+'''        <tr>
165 <th>Column v1</th>
166 <th>Column v2</th>
167 <th>Version 1</th>
168 <th>Version 2</th>
169 </tr>\n'''
170 elif mode == 't':
171 out = out+"Column v1\tColumn v2\tVersion 1\tVersion 2\n"
172 f_out.write(out.encode('utf8'))
173 #print '# Searching...'
174 #print '# Comparing...'
175 for i in range(0,54):
```

```
176 if r2[i].value != r1[index_convert(i)].value:
177 #compare_print(j, v1_row[index_convert(j)].value, v2_row[j].value)
178 compare_print(mode, r2[0].value, r1[index_convert(i)].value,
    ↪  r2[i].value, workbook.split('/')[-1], index_convert(i), i, f_out)
179 if mode == 'r' or mode == 'j':
180 f_out.write('  </table></div><br>\n')
181 elif mode == 't' or mode == 'u':
182 f_out.write("\n")
183
184 def write_removed(v2, col, row, f_out, mode):
185 if mode == 'r' or mode == 'j':
186 f_out.write('''
187 <h3>Columns invalidated by %s</h3>
188 <table about="Version2">
189 '''%(v2.split('/')[-1]))
190 elif mode == 't':
191 f_out.write("\nColumns invalidated by %s\n"%(v2.split('/')[-1]))
192
193 print "Removed Column"
194 for i in col:
195 v1_value = labels.get(i, "")
196 if mode == 'r':
197 out = u'''        <tr resource="InvlidateChange%i"
    ↪  rev="vo:invalidatedBy" typeof="vo:InvalidateChange">
198 <td resource="Attribute%i" rev="vo:Undergoes"
    ↪  typeof="vo:Attribute">%i</td>
199 <td about="Attribute%i"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
200 <span about="Version1" property="vo:hasAttribute"
    ↪  resource="Attribute%i"/>
201 </tr>
202 '''%(i, i, i, i, v1_value, i)
203 elif mode == 'j':
204 out = u'''        <tr id="InvlidateChange%i"
    ↪  about="InvlidateChange%i">
205 <td>%i</td>
```

```
206 <td>%s</td>
207 <script type="application/ld+json">
208 '''%(i, i, i, v1_value)
209 elif mode == 't':
210 out = u"%i\t%s\n"%(i, v1_value)
211 elif mode == 'u':
212 out = u"""<http://example.com/NG/Version1> vo:hasAttribute
    ↪ <http://example.com/NG/Version1/Column%s> .
213 <http://example.com/NG/Version1/%s> vo:undergoes
    ↪ <http://example.com/Changelog#InvalidateChange%i> .
214 <http://example.com/Changelog#InvalidateChange%i> a
    ↪ vo:InvalidateChange ;
215 vo:invalidatedBy <http://example.com/NG/Version2> .
216
217 """%(i, i, i, i)
218 f_out.write(out.encode('utf8'))
219 if mode == 'j':
220 json1 = {
221 "@context":context,
222 "@type":"vo:Attribute" ,
223 "@id":"".join(["http://ngdb.com/v1/Attribute", str(i)]) ,
224 "label": v1_value,
225 "undergoes":"".join([host, "InvalidateChange", str(i)]) ,
226 "@reverse" :    { "hasAttribute" : "Version1" }
227 }
228 json2 = {
229 "@context":context,
230 "@type":"vo:InvalidateChange" ,
231 "@id": "".join([host, "InvalidateChange", str(i)]) ,
232 "invalidatedBy"  :   "Version2"
233 }
234 json.dump([json1, json2], f_out, indent=4, sort_keys=True)
235 f_out.write('''
236 </script>
237 </tr>
238 ''')
```

```
239
240
241 if mode == 'r' or mode == 'j':
242 f_out.write('''      </table>
243 <h3>Rows invalidated by %s</h3>
244 <table about="Version2">
245 '''%(v2.split('/')[-1]))
246 elif mode == 't':
247 f_out.write("\nRows invalidated by %s\n"%(v2.split('/')[-1]))
248 elif mode == 'u':
249 f_out.write("\n")
250
251 print "Removed Row"
252 workbook_name = ''
253 for i, j in sorted(row, key=lambda x: x[0]):
254 if workbook_name != j:
255 workbook_name = j
256 v1_workbook = xlrd.open_workbook(workbook_name)
257 v1_sheet = v1_workbook.sheet_by_index(0)
258 v1_col = v1_sheet.col(0)
259 v1_col = [k.value for k in v1_col]
260 v1_index = v1_col.index(i)
261 if mode == 'r':
262 out = u'''        <tr resource="InvlidateChange%i"
    ↪ rev="vo:invalidatedBy" typeof="vo:InvalidateChange">
263 <td resource="Attribute%i" rev="vo:Undergoes"
    ↪ typeof="vo:Attribute">%i(%s)</td>
264 <td about="Attribute%i"
    ↪ property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
265 <span about="Version1" property="vo:hasAttribute"
    ↪ resource="Attribute%i"/>
266 </tr>
267 '''%(v1_index, v1_index, v1_index, workbook_name.split('/')[-1],
    ↪ v1_index, i, v1_index)
268 elif mode == 'j':
```

```
269 out = u'''        <tr id="InvlidateChange%i"
    ↪ about="InvlidateChange%i">
270 <td>%i(%s)</td>
271 <td>%s</td>
272 <script type="application/ld+json">
273 '''%(v1_index, v1_index, v1_index, workbook_name.split('/')[-1], i)
274 elif mode == 't':
275 out = u"%i(%s)\t%s\n"%(v1_index, workbook_name.split('/')[-1], i)
276 elif mode == 'u':
277 out = u"""<http://example.com/NG/Version1> vo:hasAttribute
    ↪ <http://example.com/NG/Version1/%s> .
278 <http://example.com/NG/Version1/%s> vo:undergoes
    ↪ <http://example.com/Changelog#InvalidateChange%s> .
279 <http://example.com/Changelog#InvalidateChange%s> a
    ↪ vo:InvalidateChange ;
280 vo:invalidatedBy <http://example.com/NG/Version2> .
281
282 """%(i, i, i, i)
283 f_out.write(out.encode('utf8'))
284 if mode == 'j':
285 json1 = {
286 "@context":context,
287 "@type":"vo:Attribute" ,
288 "@id":"".join(["http://ngdb.com/v1/Attribute", str(i)]) ,
289 "label": str(i),
290 "undergoes":"".join([host, "InvalidateChange", str(i)]) ,
291 "@reverse" :    { "hasAttribute" : "Version1" }
292 }
293 json2 = {
294 "@context":context,
295 "@type":"vo:InvalidateChange" ,
296 "@id": "".join([host, "InvalidateChange", str(i)]) ,
297 "invalidatedBy"  :   "Version2"
298 }
299 json.dump([json1, json2], f_out, indent=4, sort_keys=True)
300 f_out.write('''
```

```
301 </script>
302 </tr>
303 ''')
304 if mode == 'r' or mode == 'j':
305 f_out.write('''      </table>
306
307 ''')
308 elif mode == 't' or mode == 'u':
309 f_out.write("\n")
310
311
312 def write_added(v2, col, row, f_out, mode):
313 if mode == 'r' or mode == 'j':
314 f_out.write('''
315 <h3>Columns added by %s</h3>
316 <table about="Version1" rel="vo:absentFrom">
317 '''%(v2.split('/')[-1]))
318 elif mode == 't':
319 f_out.write("\nColumns added by %s\n\n"%(v2.split('/')[-1]))
320
321 print "Added Column"
322 for i in col:
323 print i#, v2_value
324 if mode == 'r':
325 f_out.write('''        <tr about="AddChange%i" typeof="vo:AddChange">
326 <td property="vo:resultsIn" resource="Attribute%i"
    ↪ typeof="vo:Attribute">%i</td>
327 <td about="Attribute%i"
    ↪ property="http://www.w3.org/2000/01/rdf-schema#label"></td>
328 <span about="Version2" property="vo:hasAttribute"
    ↪ resource="Attribute%i"/>
329 </tr>
330 '''%(i, i, i, i, i))
331 elif mode == 'j':
332 f_out.write('''        <tr id="AddChange%i" about="v2:Attribute%i">
333 <td>%i</td>
```

```
<td></td>
<script type="application/ld+json">
'''%(i, i, i))
json1 = {
"@context":context,
"@type":"vo:AddChange" ,
"@id": "".join([host, "AddChange", str(i)]) ,
"resultsIn" :   "".join([ "http://ngdb.com/v2/Attribute", str(i)]),
"@reverse"  :   { "absentFrom": "Version1" }
}
json2 = {
"@context":context,
"@type":"vo:Attribute" ,
"@id":"".join(["http://ngdb.com/v2/Attribute", str(i)]) ,
"label":"" ,
"@reverse" :    { "hasAttribute" : "Version2" }
}
json.dump([json1, json2], f_out, indent=4, sort_keys=True)
f_out.write('''
</script>
</tr>
''')
elif mode == 't':
f_out.write("%i\t\n"%(i))
elif mode == 'u':
f_out.write("""<http://example.com/NG/Version1> vo:absentFrom
 ↪  <http://example.com/Changelog#AddChange%i> .
<http://example.com/Changelog#AddChange%i> a vo:AddChange ;
vo:resultsIn <http://example.com/NG/Version2/Column%s> .
<http://example.com/NG/Version2> vo:hasAttribute
 ↪  <http://example.com/NG/Version2/Column%s> .

"""%(i, i, i, i))
if mode == 'r' or mode == 'j':
f_out.write('''      </table>
<h3>Rows added by %s</h3>
```

```
368 <table about="Version1" rel="vo:absentFrom">
369 '''%(v2.split('/')[-1]))
370 elif mode == 't':
371 f_out.write("\nRows added by %s\n\n"%(v2.split('/')[-1]))
372 elif mode == 'u':
373 f_out.write("\n")
374
375 print "Added Row"
376 for i, j in row:#i is the id, j is the file
377 if mode == 'r':                          #print i,
    ↪ v2_sheet.cell(i,0).value
378 out = u'''        <tr about="AddChange%i" typeof="vo:AddChange">
379 <td property="vo:resultsIn" resource="Attribute%i"
    ↪ typeof="vo:Attribute">%i</td>
380 <td about="Attribute%i"
    ↪ property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
381 <span about="Version2" property="vo:hasAttribute"
    ↪ resource="Attribute%i"/>
382 </tr>
383 '''%(i, i, i, i, j, i)
384 elif mode == 'j':
385 out = u'''        <tr id="AddChange%i" about="v2:Attribute%i">
386 <td>%i</td>
387 <td property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
388 <script type="application/ld+json">
389 '''%(i, i, i, j)
390 elif mode == 't':
391 out = u"%i\t%s\n"%(i, j)
392 elif mode == 'u':
393 out = u"""<http://example.com/NG/Version1> vo:absentFrom
    ↪ <http://example.com/Changelog#AddChange%i> .
394 <http://example.com/Changelog#AddChange%i> a vo:AddChange ;
395 vo:resultsIn <http://example.com/NG/Version2/%s> .
396 <http://example.com/NG/Version2> vo:hasAttribute
    ↪ <http://example.com/NG/Version2/%s> .
397
```

```
398 """%(i, i, i, i)
399 f_out.write(out.encode('utf8'))
400 if mode == 'j':
401 json1 = {
402 "@context":context,
403 "@type":"vo:AddChange" ,
404 "@id": "".join([host, "AddChange", str(i)]) ,
405 "resultsIn" :   "".join([ "http://ngdb.com/v2/Attribute", str(i)]),
406 "@reverse"  :   { "absentFrom": "Version1" }
407 }
408 json2 = {
409 "@context":context,
410 "@type":"vo:Attribute" ,
411 "@id":"".join(["http://ngdb.com/v2/Attribute", str(i)]) ,
412 "label": j ,
413 "@reverse" :    { "hasAttribute" : "Version2" }
414 }
415 json.dump([json1, json2], f_out, indent=4, sort_keys=True)
416 f_out.write('''
417 </script>
418 </tr>
419 ''')
420
421 if mode == 'r' or mode == 'j':
422 f_out.write('''      </table>
423 ''')
424 elif mode == 't' or mode == 'u':
425 f_out.write("\n")
426
427 def write_header(f_out, mode):
428 if mode == 'j' or mode == 'r':
429 f_out.write('''<html>
430 <head>
431 </head>
```

```
<body vocab="http://www.w3.org/nw/prov#" prefix="vo:
 ↪  https://orion.tw.rpi.edu/~blee/VersionOntology.owl# v1:
 ↪  http://ngdb.com/v1/ v2: http://ngdb.com/v2/">
'''')
if mode == 'j':
f_out.write('''  <script type="application/ld+json">
''')
json1 = {
"@context":context,
"@type":"vo:Version",
"@id":"Version1",
"label":"ngdbv1"
}
json2 = {
"@context":context,
"@type":"vo:Version",
"@id":"Version2",
"label":"DB_final-55-7262_2015_03_08.xlsx"
}
json.dump([json1,json2], f_out, indent=4, sort_keys=True)
f_out.write("\n  </script>\n")
if mode == 'u':
f_out.write("""@prefix vo:
 ↪  <http://orion.tw.rpi.edu/~blee/VersionOntology.owl#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.com/NG/Version1> a vo:Version ;
skos:prefLabel "ngdbv1" .

<http://example.com/NG/Version2> a vo:Version ;
skos:prefLabel "DB_final-55-7262_2015_03_08.xlsx" .

```

```
465 """)
466
467 def write_footer(f_out, mode):
468 if mode == 'r':
469 f_out.write('</body>\n</html>')
470
471 def get_indicator_map(excel_files):
472 indicator_map = {}
473 for excel_file in excel_files:
474 print 'Importing: ' + excel_file
475 file_workbook = xlrd.open_workbook(excel_file)
476 file_sheet = file_workbook.sheet_by_index(0)
477 indicators = file_sheet.col(0)
478 for i in range(4, file_sheet.nrows):
479 indicator_map[indicators[i].value] = excel_file
480 return indicator_map
481
482 def compare(v1s, v2, fn_out, mode):
483 indicator_map = get_indicator_map(v1s)
484 i_keys = indicator_map.keys()
485 v2_workbook = xlrd.open_workbook(v2)
486 f_out = open(fn_out, 'w')
487
488 v2_sheet = v2_workbook.sheet_by_index(0)
489 v2_keys = [i.value for i in v2_sheet.col(0)]
490
491 converted = [index_convert(i) for i in range(0,54)]
492 new_col = [i for i in range(0, v2_sheet.ncols) if index_convert(i) ==
    ↪ -1]
493 new_row = [(i, v2_sheet.cell(i,0).value) for i in range(3,
    ↪ v2_sheet.nrows) if v2_sheet.cell(i,0).value not in i_keys]
494 old_col = [i for i in range(0,194) if i not in converted]
495 old_row = [(i, indicator_map.get(i, None)) for i in i_keys if i not
    ↪ in v2_keys]
496
497 write_header(f_out, mode)
```

```
498 write_added(v2, new_col, new_row, f_out, mode)
499 write_removed(v2, old_col, old_row, f_out, mode)
500
501 if mode == 'r' or mode == 'j':
502 f_out.write('''
503 <h3>Change Log</h3>
504 ''')
505 elif mode == 't':
506 f_out.write("Change Log\n")
507
508 workbook_name = ''
509 for i in range(3,v2_sheet.nrows):
510 v2_row = v2_sheet.row(i)
511 #workbook_name = v1_file
512 if v2_row[0].value in [j for i, j in new_row] or v2_row[0].value in
    ↪  [i for i, j in old_row]:
513 continue
514 if workbook_name == indicator_map.get(v2_row[0].value, None):
515 pass
516 else:
517 workbook_name = indicator_map.get(v2_row[0].value, None)
518 v1_workbook = xlrd.open_workbook(workbook_name)
519 v1_sheet = v1_workbook.sheet_by_index(0)
520 v1_col = v1_sheet.col(0)
521 v1_col = [j.value for j in v1_col]
522 #print v2_row[0].value
523 v1_index = v1_col.index(v2_row[0].value)
524 v1_row = v1_sheet.row(v1_index)
525 write_modify(v1_row, v2_row, workbook_name, f_out, mode)
526
527 write_footer(f_out, mode)
528 f_out.close()
529
530 if __name__ == "__main__":
531 if '-json' in sys.argv:
532 mode = 'j'
```

```
533 out_name = 'changelog_json.html'
534 elif '-rdfa' in sys.argv:
535 mode = 'r'
536 out_name = 'changelog_test.html'
537 elif '-txt' in sys.argv:
538 mode = 't'
539 out_name = 'changelog.txt'
540 elif '-ttl' in sys.argv:
541 mode = 'u'
542 out_name = 'changelog.ttl'
543
544 v2_dir = join(separator, 'data', 'NGdata', 'v2')
545 v1_dir = join(separator, 'data', 'NGdata', 'v1')
546
547 excel_files = glob.glob("/data/NGdata/v1/DB_HE_6733.xlsx")
    ↪ #join(v1_dir, '*.xlsx'))
548
549 v1_file = join(v1_dir, 'America_906.xlsx')
550 v2_file = join(v2_dir, 'DB_final-55-7262_2015_03_08.xlsx')
551
552 compare(excel_files, v2_file, out_name, mode)
```

APPENDIX B
NOBLE GAS CHANGE LOG GENERATOR VERSION 2 TO 3

```
from os.path import join, dirname, abspath, isfile
from os import sep as separator
import xlrd, sys, json
import glob
import re


def index_convert(index1):
return index1

def test_alignment():
for i in range(0, 54):
print 'version2: {:5} version1: {:5}'.format(i, index_convert(i))

def compare_print(mode, key, val1, val2, v1_file, v1_index = 0,
    v2_index = 0, changelog = None):
if changelog:
if mode == 'r':
out = u'''        <tr  about="Change{}{}" typeof="vo:ModifyChange">
<td align="right" rev="vo:Undergoes" resource="v2:Attribute{}{}v2"
    typeof="vo:Attribute">{:2}({})</td>
<td property="vo:resultsIn" resource="v3:Attribute{}{}v3"
    typeof="vo:Attribute">{:2}</td>
<td>{:>10}</td>
<td>{:>10}</td>
<span about="Version2" property="vo:hasAttribute"
    resource="v2:Attribute{}{}v2"></span>
<span about="Version3" property="vo:hasAttribute"
    resource="v3:Attribute{}{}v3"></span>
```

```
25 </tr>\n'''.format(key, v2_index, key, v1_index, v1_index, v1_file,
    ↪ key, v2_index, v2_index, val1, val2, key, v1_index, key,
    ↪ v2_index)
26 elif mode == 'j':
27 out = u'''        <tr  id="ModifyChange{}{}">
28 <td align="right">{:2}</td>
29 <td>{:2}</td>
30 <td>{:>10}</td>
31 <td>{:>10}</td>
32 <script type="application/ld+json">\n'''.format(key, v2_index,
    ↪ v1_index, v2_index, val1, val2)
33 elif mode == 't':
34 out = u"{:2}\t{:2}\t{:>10}\t{:>10}\n".format(v1_index, v2_index,
    ↪ val1, val2)
35 elif mode == 'u':
36 out = u"""<http://example.com/NG/Version2> vo:hasAttribute
    ↪ <http://example.com/NG/Version2/%s> ;
37 vo:hasAttribute <http://example.com/NG/Version2/Column%i> .
38 <http://example.com/NG/Version2/%s> a vo:Attribute ;
39 vo:undergoes <http://example.com/Changelog#ModifyChange%s%i> .
40 <http://example.com/NG/Version2/Column%i> a vo:Attribute ;
41 vo:undergoes <http://example.com/Changelog#ModifyChange%s%i> .
42 <http://example.com/Changelog#ModifyChange%s%i> a vo:ModifyChange ;
43 vo:resultsIn <http://example.com/NG/Version3/%s> ;
44 vo:resultsIn <http://example.com/NG/Version3/Column%i> .
45 <http://example.com/NG/Version3> vo:hasAttribute
    ↪ <http://example.com/NG/Version3/%s> ;
46 vo:hasAttribute <http://example.com/NG/Version3/Column%i> .
47
48 """%(key, v1_index, key, key, v2_index, v1_index, key, v2_index, key,
    ↪ v2_index, key, v2_index, key, v2_index)
49 changelog.write(out.encode('utf8'))
50 if mode == 'j':
51 json1 = {
52 "@context":context,
53 "@type":"vo:Attribute" ,
```

```
54 "@id":"".join(["http://ngdb.com/v2/Attribute", key, str(v1_index)]) ,
55 "label":key ,
56 "undergoes":"".join([host, "ModifyChange", key, str(v2_index)]) ,
57 "@reverse" :    { "hasAttribute" : "Version2" }
58 }
59 json2 = {
60 "@context":context,
61 "@type":"vo:ModifyChange",
62 "@id":"".join([host, "ModifyChange", key, str(v2_index)]) ,
63 "resultsIn":"".join(["http://ngdb.com/v3/Attribute", key,
   ↪  str(v2_index)])
64 }
65 json3 = {
66 "@context":context,
67 "@type":"vo:Attribute" ,
68 "@id":"".join(["http://ngdb.com/v3/Attribute", key, str(v2_index)]) ,
69 "label":key ,
70 "@reverse" :    { "hasAttribute" : "Version3" }
71 }
72 json.dump([json1, json2, json3], changelog, indent=4, sort_keys=True)
73 changelog.write('''
74 </script>
75 </tr>
76 ''')
77 else:
78 print '{:5}  version2: {:10} version3: {:10}'.format(key, val1, val2)
79
80 labels = {}
81
82 context = "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld"
83 host =
   ↪  "http://orion.tw.rpi.edu/~blee/provdist/NobleGas/changelog_json.html#"
84 #test_alignment()
85
86
87 #print v2_row[0].value
```

```
88  #print indicator_map[v2_row[0].value]
89
90  #v1_workbook = xlrd.open_workbook(v1_file)
91  #v1_sheet = v1_workbook.sheet_by_index(0)
92  #v1_row = v1_sheet.row(4)
93
94  def write_modify(r1, r2, workbook, f_out, mode):
95  if mode == 'r':
96  out = u'''  <div about="Version2" rel="vo:hasAttribute">
97  <div resource="v3:%s" typeof="vo:Attribute">
98  <span style="font-weight:bold"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">%s</span>
99  <table rel="vo:Undergoes">
100 '''%(r2[0].value, r2[0].value)
101 elif mode == 'j':
102 out = u'''
103 <div about="v3:%s">
104 <span style="font-weight:bold"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">%s</span>
105 <table>
106 '''%(r2[0].value, r2[0].value)
107 elif mode == 't':
108 out = u"%s\n"%(r2[0].value)
109 elif mode == 'u':
110 out = u""
111
112 if mode == 'r' or mode == 'j':
113 out = out+'''        <tr>
114 <th>Column v2</th>
115 <th>Column v3</th>
116 <th>Version 2</th>
117 <th>Version 3</th>
118 </tr>\n'''
119 elif mode == 't':
120 out = out+"Column v2\tColumn v3\tVersion 2\tVersion 3\n"
121 f_out.write(out.encode('utf8'))
```

```
122 #print '# Searching...'
123 #print '# Comparing...'
124 for i in range(0,54):
125 if r2[i].value != r1[i].value:
126 #compare_print(j, v1_row[index_convert(j)].value, v2_row[j].value)
127 compare_print(mode, r2[0].value, r1[i].value, r2[i].value,
    ↪  workbook.split('/')[-1], i, i, f_out)
128 if mode == 'r' or mode == 'j':
129 f_out.write('  </table></div><br>\n')
130 elif mode == 't' or mode == 'u':
131 f_out.write("\n")
132
133 def write_removed(v2, col, row, f_out, mode):
134 if mode == 'r' or mode == 'j':
135 f_out.write('''
136 <h3>Columns invalidated by %s</h3>
137 <table about="Version2">
138 '''%(v2.split('/')[-1]))
139 elif mode == 't':
140 f_out.write("\nColumns invalidated by %s\n"%(v2.split('/')[-1]))
141
142 print "Removed Column"
143 for i in col:
144 v1_value = labels.get(i, "")
145 if mode == 'r':
146 out = u'''        <tr resource="InvlidateChange%i"
    ↪  rev="vo:invalidatedBy" typeof="vo:InvalidateChange">
147 <td resource="Attribute%i" rev="vo:Undergoes"
    ↪  typeof="vo:Attribute">%i</td>
148 <td about="Attribute%i"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
149 <span about="Version1" property="vo:hasAttribute"
    ↪  resource="Attribute%i"/>
150 </tr>
151 '''%(i, i, i, i, v1_value, i)
152 elif mode == 'j':
```

```
153 out = u'''        <tr id="InvlidateChange%i"
    ↪ about="InvlidateChange%i">
154 <td>%i</td>
155 <td>%s</td>
156 <script type="application/ld+json">
157 '''%(i, i, i, v1_value)
158 elif mode == 't':
159 out = u"%i\t%s\n"%(i, v1_value)
160 elif mode == 'u':
161 out = u"""<http://example.com/NG/Version2> vo:hasAttribute
    ↪ <http://example.com/NG/Version2/Column%i> .
162 <http://example.com/NG/Version2/Column%i> vo:undergoes
    ↪ <http://example.com/Changelog#InvalidateChange%i> .
163 <http://example.com/Changelog#InvalidateChange%i> a
    ↪ vo:InvalidateChange ;
164 vo:invalidatedBy <http://example.com/NG/Version3> .
165
166 """%(i, i, i, i)
167 f_out.write(out.encode('utf8'))
168 if mode == 'j':
169 json1 = {
170 "@context":context,
171 "@type":"vo:Attribute" ,
172 "@id":"".join(["http://ngdb.com/v2/Attribute", str(i)]) ,
173 "label": v1_value,
174 "undergoes":"".join([host, "InvalidateChange", str(i)]) ,
175 "@reverse" :    { "hasAttribute" : "Version2" }
176 }
177 json2 = {
178 "@context":context,
179 "@type":"vo:InvalidateChange" ,
180 "@id": "".join([host, "InvalidateChange", str(i)]) ,
181 "invalidatedBy"  :   "Version3"
182 }
183 json.dump([json1, json2], f_out, indent=4, sort_keys=True)
184 f_out.write('''
```

```
185 </script>
186 </tr>
187 ''')
188
189
190 if mode == 'r' or mode == 'j':
191 f_out.write('''      </table>
192 <h3>Rows invalidated by %s</h3>
193 <table about="Version2">
194 '''%(v2.split('/')[-1]))
195 elif mode == 't':
196 f_out.write("\nRows invalidated by %s\n"%(v2.split('/')[-1]))
197 elif mode == 'u':
198 f_out.write("\n")
199
200 print "Removed Row"
201 for i, j in sorted(row):#i is row #, j is row id
202 if mode == 'r':
203 out = u'''        <tr resource="InvlidateChange%s"
    ↪  rev="vo:invalidatedBy" typeof="vo:InvalidateChange">
204 <td resource="Attribute%s" rev="vo:Undergoes"
    ↪  typeof="vo:Attribute">%i</td>
205 <td about="Attribute%s"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
206 <span about="Version2" property="vo:hasAttribute"
    ↪  resource="Attribute%s"/>
207 </tr>
208 '''%(j, j, i, j, j, j)
209 elif mode == 'j':
210 out = u'''        <tr id="InvlidateChange%s"
    ↪  about="InvlidateChange%s">
211 <td>%i</td>
212 <td>%s</td>
213 <script type="application/ld+json">
214 '''%(j, j, i,  j)
215 elif mode == 't':
```

```
216 out = u"%i\t%s\n"%(i, j)
217 elif mode == 'u':
218 out = u"""<http://example.com/NG/Version2> vo:hasAttribute
    ↪ <http://example.com/NG/Version2/%s> .
219 <http://example.com/NG/Version2/%s> vo:undergoes
    ↪ <http://example.com/Changelog#InvalidateChange%s> .
220 <http://example.com/Changelog#InvalidateChange%s> a
    ↪ vo:InvalidateChange ;
221 vo:invalidatedBy <http://example.com/NG/Version2> .
222
223 """%(j, j, j, j)
224 f_out.write(out.encode('utf8'))
225 if mode == 'j':
226 json1 = {
227 "@context":context,
228 "@type":"vo:Attribute" ,
229 "@id":"".join(["http://ngdb.com/v2/Attribute", j]) ,
230 "label": j,
231 "undergoes":"".join([host, "InvalidateChange", j]) ,
232 "@reverse" :    { "hasAttribute" : "Version2" }
233 }
234 json2 = {
235 "@context":context,
236 "@type":"vo:InvalidateChange" ,
237 "@id": "".join([host, "InvalidateChange", j]) ,
238 "invalidatedBy"  :   "Version3"
239 }
240 json.dump([json1, json2], f_out, indent=4, sort_keys=True)
241 f_out.write('''
242 </script>
243 </tr>
244 ''')
245 if mode == 'r' or mode == 'j':
246 f_out.write('''      </table>
247
248 ''')
```

```
249 elif mode == 't' or mode == 'u':
250 f_out.write("\n")
251
252
253 def write_added(v2, col, row, f_out, mode):
254 if mode == 'r' or mode == 'j':
255 f_out.write('''
256 <h3>Columns added by %s</h3>
257 <table about="Version2" rel="vo:absentFrom">
258 '''%(v2.split('/')[-1]))
259 elif mode == 't':
260 f_out.write("\nColumns added by %s\n\n"%(v2.split('/')[-1]))
261
262 print "Added Column"
263 for i in col:
264 print i#, v2_value
265 if mode == 'r':
266 f_out.write('''        <tr about="AddChange%i" typeof="vo:AddChange">
267 <td property="vo:resultsIn" resource="Attribute%i"
    ↪ typeof="vo:Attribute">%i</td>
268 <td about="Attribute%i"
    ↪ property="http://www.w3.org/2000/01/rdf-schema#label"></td>
269 <span about="Version3" property="vo:hasAttribute"
    ↪ resource="Attribute%i"/>
270 </tr>
271 '''%(i, i, i, i, i))
272 elif mode == 'j':
273 f_out.write('''        <tr id="AddChange%i" about="v2:Attribute%i">
274 <td>%i</td>
275 <td></td>
276 <script type="application/ld+json">
277 '''%(i, i, i))
278 json1 = {
279 "@context":context,
280 "@type":"vo:AddChange" ,
281 "@id": "".join([host, "AddChange", str(i)]) ,
```

```
"resultsIn" :   "".join([ "http://ngdb.com/v3/Attribute", str(i)]),
"@reverse"  :   { "absentFrom": "Version2" }
}
json2 = {
"@context":context,
"@type":"vo:Attribute" ,
"@id":"".join(["http://ngdb.com/v3/Attribute", str(i)]) ,
"label":"" ,
"@reverse" :    { "hasAttribute" : "Version3" }
}
json.dump([json1, json2], f_out, indent=4, sort_keys=True)
f_out.write('''
</script>
</tr>
''')
elif mode == 't':
f_out.write("%i\t\n"%(i))
elif mode == 'u':
f_out.write("""<http://example.com/NG/Version2> vo:absentFrom
 ↪  <http://example.com/Changelog#AddChange%i> .
<http://example.com/Changelog#AddChange%i> a vo:AddChange ;
vo:resultsIn <http://example.com/NG/Version3/Column%s> .
<http://example.com/NG/Version3> vo:hasAttribute
 ↪  <http://example.com/NG/Version3/Column%s> .

"""%(i, i, i, i))
if mode == 'r' or mode == 'j':
f_out.write('''      </table>
<h3>Rows added by %s</h3>
<table about="Version2" rel="vo:absentFrom">
'''%(v2.split('/')[-1]))
elif mode == 't':
f_out.write("\nRows added by %s\n\n"%(v2.split('/')[-1]))
elif mode == 'u':
f_out.write("\n")

```

```
316 print "Added Row"
317 for i, j in row:#i is the row #, j is the id
318 if mode == 'r':                          #print i,
    ↪ v2_sheet.cell(i,0).value
319 out = u'''        <tr about="AddChange%s" typeof="vo:AddChange">
320 <td property="vo:resultsIn" resource="Attribute%s"
    ↪ typeof="vo:Attribute">%i</td>
321 <td about="Attribute%s"
    ↪ property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
322 <span about="Version3" property="vo:hasAttribute"
    ↪ resource="Attribute%s"/>
323 </tr>
324 '''%(j, j, i, j, j, j)
325 elif mode == 'j':
326 out = u'''        <tr id="AddChange%s" about="v3:Attribute%s">
327 <td>%i</td>
328 <td property="http://www.w3.org/2000/01/rdf-schema#label">%s</td>
329 <script type="application/ld+json">
330 '''%(j, j, i, j)
331 elif mode == 't':
332 out = u"%i\t%s\n"%(i, j)
333 elif mode == 'u':
334 out = u"""<http://example.com/NG/Version2> vo:absentFrom
    ↪ <http://example.com/Changelog#AddChange%s> .
335 <http://example.com/Changelog#AddChange%s> a vo:AddChange ;
336 vo:resultsIn <http://example.com/NG/Version3/%s> .
337 <http://example.com/NG/Version3> vo:hasAttribute
    ↪ <http://example.com/NG/Version3/%s> .
338
339 """%(j, j, j, j)
340 f_out.write(out.encode('utf8'))
341 if mode == 'j':
342 json1 = {
343 "@context":context,
344 "@type":"vo:AddChange" ,
345 "@id": "".join([host, "AddChange", j]) ,
```

```
346 "resultsIn" :   "".join([ "http://ngdb.com/v3/Attribute", j]),
347 "@reverse"  :   { "absentFrom": "Version2" }
348 }
349 json2 = {
350 "@context":context,
351 "@type":"vo:Attribute" ,
352 "@id":"".join(["http://ngdb.com/v3/Attribute", j]) ,
353 "label": j ,
354 "@reverse" :    { "hasAttribute" : "Version2" }
355 }
356 json.dump([json1, json2], f_out, indent=4, sort_keys=True)
357 f_out.write('''
358 </script>
359 </tr>
360 ''')
361
362 if mode == 'r' or mode == 'j':
363 f_out.write('''      </table>
364 ''')
365 elif mode == 't' or mode == 'u':
366 f_out.write("\n")
367
368 def write_header(f_out, mode):
369 if mode == 'j' or mode == 'r':
370 f_out.write('''<html>
371 <head>
372 </head>
373 <body vocab="http://www.w3.org/nw/prov#" prefix="vo:
    ↪ https://orion.tw.rpi.edu/~blee/VersionOntology.owl# v2:
    ↪ http://ngdb.com/v2/ v3: http://ngdb.com/v3/">
374 ''')
375 if mode == 'j':
376 f_out.write('''  <script type="application/ld+json">
377 ''')
378 json1 = {
379 "@context":context,
```

```
"@type":"vo:Version",
"@id":"Version2",
"label":"DB_final-55-7262_2015_03_08.xlsx"
}
json2 = {
"@context":context,
"@type":"vo:Version",
"@id":"Version3",
"label":"NG_DB_final_2017_07_01.xlsx"
}
json.dump([json1,json2], f_out, indent=4, sort_keys=True)
f_out.write("\n  </script>\n")
if mode == 'u':
f_out.write("""@prefix vo:
 ↪  <http://orion.tw.rpi.edu/~blee/VersionOntology.owl#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.com/NG/Version3> a vo:Version ;
skos:prefLabel "NG_DB_final_2017_07_01.xlsx" .

<http://example.com/NG/Version2> a vo:Version ;
skos:prefLabel "DB_final-55-7262_2015_03_08.xlsx" .

""")

def write_footer(f_out, mode):
if mode == 'r':
f_out.write('</body>\n</html>')

def get_indicator_map(excel_files):
indicator_map = {}
for excel_file in excel_files:
```

```
415 print 'Importing: ' + excel_file
416 file_workbook = xlrd.open_workbook(excel_file)
417 file_sheet = file_workbook.sheet_by_index(0)
418 indicators = file_sheet.col(0)
419 for i in range(4, file_sheet.nrows):
420 indicator_map[indicators[i].value] = excel_file
421 return indicator_map
422
423 def compare(v1s, v2, fn_out, mode):
424 v1_workbook = xlrd.open_workbook(v1s)
425 v1_sheet = v1_workbook.sheet_by_index(0)
426 i_keys = {j.value:i for i,j in enumerate(v1_sheet.col(0)[3:],3)}
427
428 v2_workbook = xlrd.open_workbook(v2)
429 v2_sheet = v2_workbook.sheet_by_index(0)
430 v2_keys = {j.value:i for i,j in enumerate(v2_sheet.col(0)[3:],3)}
431
432 f_out = open(fn_out, 'w')
433
434 new_col = [i for i in range(0, v2_sheet.ncols) if index_convert(i) ==
    ↪ -1]
435 new_row = [(v2_keys[i], i) for i in v2_keys.keys() if i not in
    ↪ i_keys.keys()]
436 old_col = [i for i in range(0, v1_sheet.ncols) if index_convert(i) ==
    ↪ -1]
437 old_row = [(i_keys[i], i) for i in i_keys.keys() if i not in
    ↪ v2_keys.keys()]
438
439 write_header(f_out, mode)
440 write_added(v2, new_col, new_row, f_out, mode)
441 write_removed(v2, old_col, old_row, f_out, mode)
442
443 if mode == 'r' or mode == 'j':
444 f_out.write('''
445 <h3>Change Log</h3>
446 ''')
```

```
447 elif mode == 't':
448 f_out.write("Change Log\n")
449
450 workbook_name = ''
451 for i in range(3,v2_sheet.nrows):
452 v2_row = v2_sheet.row(i)
453 #workbook_name = v1_file
454 if v2_row[0].value in [j for i, j in new_row] or v2_row[0].value in
    ↪  [j for i, j in old_row]:
455 continue
456 v1_row = v1_sheet.row(i_keys[v2_row[0].value])
457 write_modify(v1_row, v2_row, workbook_name, f_out, mode)
458
459 write_footer(f_out, mode)
460 f_out.close()
461
462 if __name__ == "__main__":
463 if '-json' in sys.argv:
464 mode = 'j'
465 out_name = 'isotope2_3_json.html'
466 elif '-rdfa' in sys.argv:
467 mode = 'r'
468 out_name = 'isotope2_3_rdfa.html'
469 elif '-txt' in sys.argv:
470 mode = 't'
471 out_name = 'changelog2_3.txt'
472 elif '-ttl' in sys.argv:
473 mode = 'u'
474 out_name = 'changelog2_3.ttl'
475
476 v2_dir = join(separator, 'data', 'NGdata', 'v2')
477 v3_dir = join(separator, 'data', 'NGdata', 'v3')
478
479 v2_file = join(v2_dir, 'DB_final-55-7262_2015_03_08.xlsx')
480 v3_file = join(v3_dir, 'NG_DB_final_2017_07_01.xlsx')
481
```

```
482 compare(v2_file, v3_file, out_name, mode)
```

APPENDIX C
GLOBAL CHANGE MASTER DIRECTORY CHANGE LOG GENERATOR VERSION JUNE 12, 2012 TO VERSION 8.4.1

```
import glob, json, rdflib, re
from rdflib import URIRef, Literal, Namespace
from rdflib.namespace import RDF, SKOS

def GCMDChangeLogGenerator(GCMDfile):
GCMD = Namespace("http://gcmdservices.gsfc.nasa.gov/kms/concept/")

#GCMDfile = ['GCMD8_3.rdf', 'GCMD8_4.rdf','GCMD8_4_1.rdf']
numbers = [re.search('GCMD(.*).rdf', i).group(1).replace("_","") for
 ↪  i in GCMDfile]
print numbers
filename = 'ChangelogGCMD'+"_".join(numbers)+'.html'
output = open(filename, 'w')
 ↪  #'/home/blee/provdist/GCMD/webGCMD83_84.html', 'w')
#output = codecs.open('/home/blee/GCMD/GCMD8_3to8_4.html',
 ↪  mode='w', encoding='utf-8')

g0 = rdflib.Graph()
g0.parse(GCMDfile[0])
g1 = rdflib.Graph()
g1.parse(GCMDfile[1])

ver = [re.search('GCMD(.*).rdf', i).group(1).replace("_",".") for i
 ↪  in GCMDfile]#['8.3', '8.4']
print ver
new = g1-g0
old = g0-g1

```

```
25 #Get the date for the change notes in the new changes made in version 2
26 #This will help determine if some concepts were changed without being moved
27 #Their change notes should have a date on the same day as the new additions.
28 #This is probably a bad way of determining this.
29 date = g1.value(new.value(predicate=RDF.type, object=SKOS.Concept), SKOS.changeNote).split()[0]
30 context = "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld"
31
32 ###################
33 ###   Header   ###
34 ###################
35
36 output.write('''<html>
37 <head>
38 <link
   href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
   rel="stylesheet"
   integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdE
   crossorigin="anonymous">
39 </head>
40 <body vocab="http://www.w3.org/nw/prov#" prefix="gcmd: http://gcmdservices.gsfc.nasa.gov/kms/concept/">
41 <h2 property="http://purl.org/dc/terms/title">
42 <span about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" property="http://www.w3.org/2000/01/rdf-schema#label">%s</span> to
43 <span about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" property="http://www.w3.org/2000/01/rdf-schema#label">%s</span>
44 '''%(ver[0], GCMDfile[0], ver[1], GCMDfile[1]))
45
46 output.write('''      <script type="application/ld+json">
```

```
47 [
48 {
49 "@context" : "%s" ,
50 "type"        :        "vo:Version" ,
51 "id"        :        "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
   ↪  ,
52 "label" :       "%s"
53 },
54 {
55 "@context" : "%s",
56 "type"  :       "vo:Version" ,
57 "id"    :
   ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" ,
58 "label" :       "%s"
59 }
60 ]
61 </script>
62 </h2>
63 '''%(context, ver[0], GCMDfile[0], context, ver[1], GCMDfile[1]))
64
65 ##################
66 ###   ADDED   ###
67 ##################
68
69 output.write('''
70 <h3>Concepts added to %s</h3>
71 <table
   ↪  about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
   ↪  class="table table-striped">
72 <tr>
73 <th>Link</th>
74 <th>Concept</th>
75 <th>Change Note</th>
76 </tr>
77 '''%(GCMDfile[1], ver[1]))
78
```

```
79  c = 0
80
81  for i in new.subjects(RDF.type, SKOS.Concept):
82  changeNote = "<br>\n              ".join(g1.objects(i,
    ↪  SKOS.changeNote))
83  output.write((u'''        <tr id="AddChange%i" about="%s?version=%s">
84  <td>
85  <a href="%s?version=%s">Link</a>
86  </td>
87  <td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
88  <td property="http://www.w3.org/2004/02/skos/core#changeNote">%s</td>
89  '''%(c, str(i), ver[1], str(i), ver[1], g1.value(i, SKOS.prefLabel),
    ↪  changeNote)).encode('utf8'))
90  output.write((u'''
91  <script type="application/ld+json">
92  [
93  {
94  "@context" : "%s" ,
95  "type"        :        "vo:AddChange" ,
96  "id"        :        "this:AddChange%i" ,
97  "resultsIn" :        "gcmd:%s?version=%s" ,
98  "@reverse"  :        { "absentFrom":
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
99  },
100 {
101 "@context" : "%s" ,
102 "type"        :        "vo:Attribute" ,
103 "id"        :        "gcmd:%s?version=%s" ,
104 "label" :        "%s" ,
105 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
106 }
107 ]
108 </script>
109 </tr>
```

```
110 '''%(context, c, i.split('/')[-1], ver[1], ver[0], context,
    ↪ i.split('/')[-1], ver[1], g1.value(i, SKOS.prefLabel),
    ↪ ver[1])).encode('utf8'))
111 c += 1
112
113 output.write('''      </table>
114
115 ''')
116
117 #print date
118
119 ##################
120 ### REMOVED   ###
121 ##################
122
123 output.write('''
124 <h3>Concepts removed from %s</h3>
125 <table
    ↪ about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
    ↪ class="table table-striped">
126 <tr>
127 <th>Link</th>
128 <th>Concept</th>
129 </tr>
130 '''%(GCMDfile[0], ver[0]))
131
132 c = 0
133
134 for i in old.subjects(RDF.type, SKOS.Concept):#Reverse relations due
    ↪ to ordering and structure
135 output.write((u'''        <tr id="InvlidateChange%i"
    ↪ about="%s?version=%s">
136 <td>
137 <a href="%s?version=%s">Link</a>
138 </td>
139 <td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
```

```
140 '''%(c, str(i), ver[0], str(i), ver[0], g0.value(i, SKOS.prefLabel),
    ↪  )).encode('utf8'))
141 output.write((u'''          <script type="application/ld+json">
142 [
143 {
144 "@context" : "%s" ,
145 "type"        :        "vo:Attribute" ,
146 "id"        :        "gcmd:%s?version=%s" ,
147 "label"        :        "%s" ,
148 "undergoes" :        "this:InvalidateChange%i" ,
149 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
150 },
151 {
152 "@context" : "%s" ,
153 "type"        :        "vo:InvalidateChange" ,
154 "id"        :        "this:InvalidateChange%i" ,
155 "invalidatedBy"        :        "gcmd:concept_scheme/sciencekeywords/?format=xml&v
156 }
157 ]
158 </script>
159 </tr>
160 '''%(context, i.split('/')[-1], ver[0], g0.value(i, SKOS.prefLabel),
    ↪  c, ver[0], context, c, ver[1])).encode('utf8'))
161 c += 1
162 output.write('''      </table>
163
164 ''')
165
166 ##################
167 ###   Modify   ###
168 ##################
169
170 output.write('''
171 <h3>Moved Concepts</h3>
172 <table class="table table-striped">
```

```
<tr>
<th>Link v1</th>
<th>Link v2</th>
<th>Label</th>
<th>Old Parent</th>
<th>New Parent</th>
</tr>\n
''')

c = 0

for i in g1.subjects(RDF.type, SKOS.Concept):
if (i, None, None) in g0:
b0 = g0.value(i, SKOS.broader)
b1 = g1.value(i, SKOS.broader)
if b0 != b1:
output.write((u'''        <tr id="MoveChange%i"
 ↪  about="%s?version=%s">
<td><a href=%s?version=%s>Link</a></td>
<td><a href=%s?version=%s>Link</a></td>
<td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
<td about="%s?version=%s"
 ↪  property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
<td about="%s?version=%s"
 ↪  property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
'''%(c, str(i), ver[1],
str(i), ver[0],
str(i), ver[1],
g1.value(i, SKOS.prefLabel),
b0, ver[0], g0.value(b0, SKOS.prefLabel),
b1, ver[1], g1.value(b1, SKOS.prefLabel))  ).encode('utf8'))

output.write((u'''          <script  type="application/ld+json">
[
{
"@context" : "%s" ,
```

```
206 "type"        :        "vo:Attribute" ,
207 "id"        :        "gcmd:%s?version=%s" ,
208 "label"        :        "%s" ,
209 "undergoes" :        "this:MoveChange%i" ,
210 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
211 },
212 {
213 "@context" : "%s" ,
214 "type"        :        "vo:MoveChange" ,
215 "id"        :        "this:MoveChange%i" ,
216 "resultsIn"        :        "gcmd:%s?version=%s"
217 },
218 {
219 "@context" : "%s" ,
220 "type"        :        "vo:Attribute" ,
221 "id"        :        "gcmd:%s?version=%s" ,
222 "label"        :        "%s" ,
223 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
224 }
225 ]
226 </script>
227 </tr>
228 '''%(context, i.split('/')[-1], ver[0], g0.value(i, SKOS.prefLabel),
    ↪  c, ver[0],
229 context, c, i.split('/')[-1], ver[1],
230 context, i.split('/')[-1], ver[1], g1.value(i, SKOS.prefLabel),
    ↪  ver[1])  ).encode('utf8'))
231 c += 1
232
233 output.write('''      </table>
234
235 ''')
236
237 ###################################
```

```
238 ###   NON-STRUCTURAL CHANGES   ###
239 ####################################
240
241 output.write('''
242 <h3>Non-Structural Changes</h3>
243 <table class="table table-striped">
244 <tr>
245 <th>Link v1</th>
246 <th>Link v2</th>
247 <th>Label</th>
248 <th>Change Notes</th>
249 </tr>\n
250 ''')
251
252 c = 0
253
254 for i in g1.subjects(RDF.type, SKOS.Concept):
255 if (i, None, None) in g0:
256 b0 = g0.value(i, SKOS.broader)
257 b1 = g1.value(i, SKOS.broader)
258 if b0 == b1:
259 new_note = False
260 notes = []
261 for note in g1.objects(i, SKOS.changeNote):
262 note_date = note.split()[0]
263 #print note_date
264 if note_date == date:
265 new_note = True
266 notes.append(note)
267 if new_note:
268 output.write((u'''        <tr id="ModifyChange%i"
    ↪  about="%s?version=%s">
269 <td><a href=%s?version=%s>Link</a></td>
270 <td><a href=%s?version=%s>Link</a></td>
271 <td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
272 <td property="http://www.w3.org/2004/02/skos/core#changeNote">%s</td>
```

```
273 '''%(c, str(i), ver[1],
274 str(i), ver[0],
275 str(i), ver[1],
276 g1.value(i, SKOS.prefLabel),
277 "<br>\n               ".join(notes)
278 )).encode('utf8'))
279 output.write((u'''
280 </tr>
281 '''%()).encode('utf8'))
282 c += 1
283
284 output.write('''      </table>
285
286 ''')
287 output.write("\t</body>\n</html>")
288 output.close()
289
290 if __name__ == "__main__":
291 GCMDfiles = sorted(glob.glob("*.rdf"))
292 for i in range(len(GCMDfiles)-1):
293 print "Starting",GCMDfiles[i-1],"and",GCMDfiles[i] #It's done this
    ↪ way because GCMDJun1220012 sorts to the last item
294 GCMDChangeLogGenerator([GCMDfiles[i-1],GCMDfiles[i]])
```

APPENDIX D
GLOBAL CHANGE MASTER DIRECTORY CHANGE LOG GENERATOR VERSION 8.4.1 TO 8.5

```
import glob, json, rdflib, re
from rdflib import URIRef, Literal, Namespace
from rdflib.namespace import RDF, SKOS

def GCMDChangeLogGenerator(GCMDfile):
GCMD = Namespace("http://gcmdservices.gsfc.nasa.gov/kms/concept/")
GCMD8_5 =
 ↪  Namespace("https://gcmdservices.gsfc.nasa.gov/kms/concept/")

#GCMDfile = ['GCMD8_3.rdf', 'GCMD8_4.rdf','GCMD8_4_1.rdf']
numbers = [re.search('GCMD(.*).rdf', i).group(1).replace("_","") for
 ↪  i in GCMDfile]
print numbers
filename = 'ChangelogGCMD2'+"_".join(numbers)+'.html'
output = open(filename, 'w')
 ↪  #'/home/blee/provdist/GCMD/webGCMD83_84.html', 'w')
#output = codecs.open('/home/blee/GCMD/GCMD8_3to8_4.html',
 ↪  mode='w', encoding='utf-8')

g0 = rdflib.Graph()
g0.parse(GCMDfile[0])
g1 = rdflib.Graph()
g1.parse(GCMDfile[1])

ver = [re.search('GCMD(.*).rdf', i).group(1).replace("_",".") for i
 ↪  in GCMDfile]#['8.3', '8.4']
print ver
new = rdflib.Graph()
for s, p, o in g1.triples((None, RDF.type, SKOS.Concept)):
if not (GCMD[s.split('/')[-1]], p, o) in g0:
```

```
26 new.add((s, p, o))
27 old = rdflib.Graph()
28 for s, p, o in g0.triples((None, RDF.type, SKOS.Concept)):
29 if not (GCMD8_5[s.split('/')[-1]], p, o) in g1:
30 old.add((s, p, o))
31
32 #Get the date for the change notes in the new changes made in
   ↪ version 2
33 #This will help determine if some concepts were changed without
   ↪ being moved
34 #Their change notes should have a date on the same day as the new
   ↪ additions.
35 #This is probably a bad way of determining this.
36 date = g1.value(new.value(predicate=RDF.type, object=SKOS.Concept),
   ↪ SKOS.changeNote).split()[0]
37 context = "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld"
38
39 ###################
40 ###   Header   ###
41 ###################
42
43 output.write('''<html>
44 <head>
45 <link
   ↪ href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
   ↪ rel="stylesheet"
   ↪ integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdE
   ↪ crossorigin="anonymous">
46 </head>
47 <body vocab="http://www.w3.org/nw/prov#" prefix="gcmd:
   ↪ http://gcmdservices.gsfc.nasa.gov/kms/concept/">
48 <h2 property="http://purl.org/dc/terms/title">
49 <span
   ↪ about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
   ↪ property="http://www.w3.org/2000/01/rdf-schema#label">%s</span>
   ↪ to
```

```
50 <span
   ↪ about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
   ↪ property="http://www.w3.org/2000/01/rdf-schema#label">%s</span>
51 '''%(ver[0], GCMDfile[0], ver[1], GCMDfile[1]))
52
53 output.write('''      <script type="application/ld+json">
54 [
55 {
56 "@context" : "%s" ,
57 "type"        :        "vo:Version" ,
58 "id"         :        "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
   ↪ ,
59 "label" :       "%s"
60 },
61 {
62 "@context" : "%s",
63 "type"  :       "vo:Version" ,
64 "id"    :
   ↪ "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" ,
65 "label" :       "%s"
66 }
67 ]
68 </script>
69 </h2>
70 '''%(context, ver[0], GCMDfile[0], context, ver[1], GCMDfile[1]))
71
72 ##################
73 ###   ADDED   ###
74 ##################
75
76 output.write('''
77 <h3>Concepts added to %s</h3>
78 <table
   ↪ about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
   ↪ class="table table-striped">
79 <tr>
```

```
<th>Link</th>
<th>Concept</th>
<th>Change Note</th>
</tr>
'''%(GCMDfile[1], ver[1]))

c = 0

for i in new.subjects(RDF.type, SKOS.Concept):
changeNote = "<br>\n               ".join(g1.objects(i,
 ↪  SKOS.changeNote))
output.write((u'''        <tr id="AddChange%i" about="%s?version=%s">
<td>
<a href="%s?version=%s">Link</a>
</td>
<td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
<td property="http://www.w3.org/2004/02/skos/core#changeNote">%s</td>
'''%(c, str(i), ver[1], str(i), ver[1], g1.value(i, SKOS.prefLabel),
 ↪  changeNote)).encode('utf8'))
output.write((u'''
<script type="application/ld+json">
[
{
"@context" : "%s" ,
"type"        :        "vo:AddChange" ,
"id"        :        "this:AddChange%i" ,
"resultsIn" :        "gcmd:%s?version=%s" ,
"@reverse"  :        { "absentFrom":
 ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
},
{
"@context" : "%s" ,
"type"        :        "vo:Attribute" ,
"id"        :        "gcmd:%s?version=%s" ,
"label" :        "%s" ,
```

```
"@reverse" :        { "hasAttribute" :
    "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
}
]
</script>
</tr>
'''%(context, c, i.split('/')[-1], ver[1], ver[0], context,
    i.split('/')[-1], ver[1], g1.value(i, SKOS.prefLabel),
    ver[1])).encode('utf8'))
c += 1

output.write('''      </table>

''')

#print date

##################
### REMOVED   ###
##################

output.write('''
<h3>Concepts removed from %s</h3>
<table
    about="gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s"
    class="table table-striped">
<tr>
<th>Link</th>
<th>Concept</th>
</tr>
'''%(GCMDfile[0], ver[0]))

c = 0

for i in old.subjects(RDF.type, SKOS.Concept):#Reverse relations due
    to ordering and structure
```

```
142 output.write((u'''        <tr id="InvlidateChange%i"
    ↪  about="%s?version=%s">
143 <td>
144 <a href="%s?version=%s">Link</a>
145 </td>
146 <td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
147 '''%(c, str(i), ver[0], str(i), ver[0], g0.value(i, SKOS.prefLabel),
    ↪  )).encode('utf8'))
148 output.write((u'''          <script type="application/ld+json">
149 [
150 {
151 "@context" : "%s" ,
152 "type"        :        "vo:Attribute" ,
153 "id"        :        "gcmd:%s?version=%s" ,
154 "label"        :        "%s" ,
155 "undergoes" :        "this:InvalidateChange%i" ,
156 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
157 },
158 {
159 "@context" : "%s" ,
160 "type"        :        "vo:InvalidateChange" ,
161 "id"        :        "this:InvalidateChange%i" ,
162 "invalidatedBy"        :        "gcmd:concept_scheme/sciencekeywords/?format=xml&v
163 }
164 ]
165 </script>
166 </tr>
167 '''%(context, i.split('/')[-1], ver[0], g0.value(i, SKOS.prefLabel),
    ↪  c, ver[0], context, c, ver[1])).encode('utf8'))
168 c += 1
169 output.write('''      </table>
170
171 ''')
172
173 ##################
```

```
174 ###   Modify   ###
175 ##################
176
177 output.write('''
178 <h3>Moved Concepts</h3>
179 <table class="table table-striped">
180 <tr>
181 <th>Link v1</th>
182 <th>Link v2</th>
183 <th>Label</th>
184 <th>Old Parent</th>
185 <th>New Parent</th>
186 </tr>\n
187 ''')
188
189 c = 0
190
191 for i in g1.subjects(RDF.type, SKOS.Concept):
192 i_ = GCMD[i.split('/')[-1]]
193 if (i_, None, None) in g0:
194 b0 = g0.value(i_, SKOS.broader)
195 b1 = g1.value(i, SKOS.broader)
196 if b1 != None:
197 b1_ = GCMD[b1.split('/')[-1]]
198 if b0 != b1_:
199 output.write((u'''        <tr id="MoveChange%i"
    ↪  about="%s?version=%s">
200 <td><a href=%s?version=%s>Link</a></td>
201 <td><a href=%s?version=%s>Link</a></td>
202 <td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
203 <td about="%s?version=%s"
    ↪  property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
204 <td about="%s?version=%s"
    ↪  property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
205 '''%(c, str(i), ver[1],
206 str(i), ver[0],
```

```
207 str(i), ver[1],
208 g1.value(i, SKOS.prefLabel),
209 b0, ver[0], g0.value(b0, SKOS.prefLabel),
210 b1, ver[1], g1.value(b1, SKOS.prefLabel))  ).encode('utf8'))
211
212 output.write((u'''          <script  type="application/ld+json">
213 [
214 {
215 "@context" : "%s" ,
216 "type"        :        "vo:Attribute" ,
217 "id"        :        "gcmd:%s?version=%s" ,
218 "label"        :        "%s" ,
219 "undergoes" :        "this:MoveChange%i" ,
220 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
221 },
222 {
223 "@context" : "%s" ,
224 "type"        :        "vo:MoveChange" ,
225 "id"        :        "this:MoveChange%i" ,
226 "resultsIn"        :        "gcmd:%s?version=%s"
227 },
228 {
229 "@context" : "%s" ,
230 "type"        :        "vo:Attribute" ,
231 "id"        :        "gcmd:%s?version=%s" ,
232 "label"        :        "%s" ,
233 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
234 }
235 ]
236 </script>
237 </tr>
238 '''%(context, i.split('/')[-1], ver[0], g0.value(i_, SKOS.prefLabel),
    ↪  c, ver[0],
239 context, c, i.split('/')[-1], ver[1],
```

```
240 context, i.split('/')[-1], ver[1], g1.value(i, SKOS.prefLabel),
    ↪  ver[1])  ).encode('utf8'))
241 c += 1
242
243 output.write('''      </table>
244
245 ''')
246
247 output.write('''
248 <h3>Modified Concepts</h3>
249 <table class="table table-striped">
250 <tr>
251 <th>Link v1</th>
252 <th>Link v2</th>
253 <th>Label</th>
254 </tr>\n
255 ''')
256
257 c = 0
258
259
260 ###################################
261 ###   NON-STRUCTURAL CHANGES   ###
262 ###################################
263
264 output.write('''
265 <h3>Non-Structural Changes</h3>
266 <table class="table table-striped">
267 <tr>
268 <th>Link v1</th>
269 <th>Link v2</th>
270 <th>Label</th>
271 <th>Change Notes</th>
272 </tr>\n
273 ''')
274
```

```
275 c = 0
276
277 for i in g1.subjects(RDF.type, SKOS.Concept):
278 i_ = GCMD[i.split('/')[-1]]
279 if (i_, None, None) in g0:
280 b0 = g0.value(i_, SKOS.broader)
281 b1 = g1.value(i, SKOS.broader)
282 if b1 != None:
283 b1_ = GCMD[b1.split('/')[-1]]
284 if b0 == b1_ and i != i_:
285 output.write((u'''        <tr id="NameChange%i"
    ↪  about="%s?version=%s">
286 <td><a href=%s?version=%s>Link</a></td>
287 <td><a href=%s?version=%s>Link</a></td>
288 <td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
289 '''%(c, str(i), ver[1],
290 str(i_), ver[0],
291 str(i), ver[1],
292 g1.value(i, SKOS.prefLabel)
293 )))
294 output.write((u'''          <script  type="application/ld+json">
295 [
296 {
297 "@context" : "%s" ,
298 "type"        :        "vo:Attribute" ,
299 "id"        :        "%s?version=%s" ,
300 "label"        :        "%s" ,
301 "undergoes" :        "this:NameChange%i" ,
302 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
303 },
304 {
305 "@context" : "%s" ,
306 "type"        :        "vo:ModifyChange" ,
307 "id"        :        "this:NameChange%i" ,
308 "resultsIn"        :        "%s?version=%s"
```

```
309 },
310 {
311 "@context" : "%s" ,
312 "type"        :        "vo:Attribute" ,
313 "id"        :        "%s?version=%s" ,
314 "label"        :        "%s" ,
315 "@reverse" :        { "hasAttribute" :
    ↪  "gcmd:concept_scheme/sciencekeywords/?format=xml&version=%s" }
316 }
317 ]
318 </script>
319 </tr>
320 '''%(context, i_, ver[0], g0.value(i_, SKOS.prefLabel), c, ver[0],
321 context, c, i, ver[1],
322 context, i, ver[1], g1.value(i, SKOS.prefLabel), ver[1])
    ↪  ).encode('utf8'))
323 c += 1
324
325 output.write('''      </table>
326
327 ''')
328
329 for i in g1.subjects(RDF.type, SKOS.Concept):
330 i_ = GCMD[i.split('/')[-1]]
331 if (i_, None, None) in g0:
332 b0 = g0.value(i_, SKOS.broader)
333 b1 = g1.value(i, SKOS.broader)
334 if b1 != None:
335 b1_ = GCMD[b1.split('/')[-1]]
336 if b0 == b1_:
337 new_note = False
338 notes = []
339 for note in g1.objects(i, SKOS.changeNote):
340 note_date = note.split()[0]
341 #print note_date
342 if note_date == date:
```

```
343 new_note = True
344 notes.append(note)
345 if new_note:
346 output.write((u'''        <tr id="ModifyChange%i"
    ↪  about="%s?version=%s">
347 <td><a href=%s?version=%s>Link</a></td>
348 <td><a href=%s?version=%s>Link</a></td>
349 <td property="http://www.w3.org/2004/02/skos/core#prefLabel">%s</td>
350 <td property="http://www.w3.org/2004/02/skos/core#changeNote">%s</td>
351 '''%(c, str(i), ver[1],
352 str(i), ver[0],
353 str(i), ver[1],
354 g1.value(i, SKOS.prefLabel),
355 "<br>\n              ".join(notes)
356 )).encode('utf8'))
357 output.write((u'''
358 </tr>
359 '''%()).encode('utf8'))
360 c += 1
361
362 output.write('''      </table>
363
364 ''')
365 output.write("\t</body>\n</html>")
366 output.close()
367
368 if __name__ == "__main__":
369 GCMDfiles = sorted(glob.glob("*.rdf"))
370 GCMDfiles = ["GCMD8_5.rdf", "GCMD8_4_1.rdf"]
371 for i in range(len(GCMDfiles)-1):
372 print "Starting",GCMDfiles[i-1],"and",GCMDfiles[i] #It's done this
    ↪  way because GCMDJun1220012 sorts to the last item
373 GCMDChangeLogGenerator([GCMDfiles[i-1],GCMDfiles[i]])
```

APPENDIX E
TURTLE EXTRACTOR

```
from bs4 import BeautifulSoup
import glob, rdflib, json, re

def extracting(f, d):
#f = 'ChangelogGCMD70_80.html'
#d = 'Graph'+re.search('ChangelogGCMD(.*).html', f).group(1)+'.ttl'

fp = open(f)
soup = BeautifulSoup(fp, 'html5lib')
fp.close()

print 'extracting...'
js = soup.find_all('script')
items = [item for sublist in js for item in json.loads(sublist.text)]

print 'loading...'
g = rdflib.Graph()
g.parse(data = json.dumps(items), format='json-ld')

print 'writing...'
g.serialize(destination=d, format='turtle')
print 'written'

if __name__ == "__main__":
l = glob.glob('Changelog*.html')
for i in l:
d = 'Graph'+re.search('ChangelogGCMD(.*).html', i).group(1)+'.ttl'
print "Extracting: "+i
print "\tto", d
extracting(i, d)
```

APPENDIX F
MARINE BIODIVERSITY VIRTUAL LABORATORY CLASSIFIER COMPARISON

```
import urllib

class entry:
query = None
dist = None
freq = None
tax = None
poss = None

def __lt__(self, other):
if self.query == other.query:
return self.freq < other.freq
else:
return self.query < other.query

def is_number(s):
try:
float(s)
return True
except ValueError:
return False

def file_parse(f_name):
f = open(f_name, 'r')
found = {}
for i in mock:
found[i] = [[],[],0]
ambiguous = []
fp = []
fn_count = 0
```

```
fn = []
results = {}
total = 0
if f_name.split('_')[1] == 'spingo':
if f_name == "silva_spingo":
family[4] = 'Clostridiaceae_1'
else:
family[2] = 'Bacillaceae 1'
family[4] = 'Clostridiaceae 1'
for line in f:
x = line.split('\t')
entry_id = x[0].split('|')[0]
results[entry_id] = entry()
results[entry_id].query = x[0]
results[entry_id].freq =
    ↪  int(results[entry_id].query.split('|')[-1].split(':')[1])
results[entry_id].dist = float(x[1])
results[entry_id].tax = x[2:]

total += results[entry_id].freq

if not is_number(results[entry_id].tax[-1]):
results[entry_id].poss = results[entry_id].tax[-1].split(',')
results[entry_id].tax = results[entry_id].tax[:-1]
if results[entry_id].tax[-2] != 'AMBIGUOUS':
print results[entry_id].tax[-2]

clean_tax = [x for x in results[entry_id].tax if not is_number(x)]
if len(clean_tax) < 7:
clean_tax[0:0] = ['Pass', 'Pass', 'Pass', 'Pass']
del clean_tax[5]
results[entry_id].tax = clean_tax

if clean_tax[4] == 'AMBIGUOUS':
fn_count += 1
ambiguous.append(entry_id)
```

```
elif not clean_tax[4] in family:
fp.append((entry_id, 'family'))
elif clean_tax[5] == 'AMBIGUOUS':
for i in range(len(family)):
if clean_tax[4] == family[i]:
found[mock[i]][0].append(entry_id)
elif not clean_tax[5] in genus:
fp.append((entry_id, 'genus'))
elif clean_tax[6] == 'AMBIGUOUS':
for i in range(len(genus)):
if clean_tax[5] == genus[i]:
found[mock[i]][1].append(entry_id)
elif not ' '.join(clean_tax[6].split('_')) in mock:
fp.append((entry_id, 'species'))
else:
found[' '.join(clean_tax[6].split('_'))][2] += 1

elif f_name.split('_')[1] == 'gast':
skip = False
if f_name == "rdp_gast":
family[2] = 'Bacillaceae 1'
family[4] = 'Clostridiaceae 1'
for line in f:
if not skip:
skip = True
continue
x = line.split('\t')
entry_id = x[0].split('|')[0]
results[entry_id] = entry()
results[entry_id].query = x[0]
results[entry_id].freq =
 ↪  int(results[entry_id].query.split('|')[-1].split(':')[1])
results[entry_id].dist = float(x[2])
results[entry_id].tax = x[1].split(';')

total += results[entry_id].freq
```

```
101 temp_tax = results[entry_id].tax
102
103 if len(temp_tax) < 5:
104 fn_count += 1
105 ambiguous.append(entry_id)
106 elif not temp_tax[4] in family:
107 fp.append((entry_id, 'family'))
108 elif len(temp_tax) < 6:
109 for i in range(len(family)):
110 if temp_tax[4] == family[i]:
111 found[mock[i]][0].append(entry_id)
112 elif not temp_tax[5] in genus:
113 fp.append((entry_id, 'genus'))
114 elif len(temp_tax) < 7:
115 for i in range(len(genus)):
116 if temp_tax[5] == genus[i]:
117 found[mock[i]][1].append(entry_id)
118 elif not ' '.join(results[entry_id].tax[5:7]) in mock:
119 fp.append((entry_id, 'species'))
120 else:
121 found[' '.join(results[entry_id].tax[5:7])][2] += 1
122
123 fp_file = open("fp_"+f_name+".txt", 'w')
124 for fid, frank in fp:
125 fp_file.write("\t".join([fid, str(results[fid].freq), frank,
    ↪  str(results[fid].tax)])+"\n")
126 fp_file.close()
127
128 for i in found.keys():
129 if sum([len(found[i][0]), len(found[i][1]), found[i][2]]) == 0:
130 fn.append(i)
131 print fn
132 ttl_output(f_name, fp, fn, found, results)
133 total = len(results.keys())
134 print "\t".join(["False Positives:", str(len(fp))])
135 print "\t".join(["Ambiguous:", str(len(ambiguous))])
```

```
136 print "\t".join(["Total:", str(len(results.keys()))])
137 coverage = sum([len(found[i][0])+len(found[i][1])+found[i][2] for i
    ↪ in found.keys()])
138 print "\t".join(["Coverage:", str(coverage)])
139 print "\t".join(["Percentage:", str(float(coverage)/total*100)])
140 for k in sorted(mock):
141 print "\t".join([k+":", str(len(found[k][0])), str(len(found[k][1])),
    ↪ str(found[k][2])])
142
143 return results
144
145 def ttl_output(f_name, false_positive, false_negative, found,
    ↪ results):
146 fp_file = open("fp_"+f_name+".ttl", 'w')
147 fp_file.write("""@prefix vo:
    ↪ <http://orion.tw.rpi.edu/~blee/VersionOntology.owl#> .
148 @prefix mbvl: <http://example.com/MBVL/> .
149 @prefix next: <http://example.com/MBVL/%s> .
150 @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
151 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
152 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
153 @prefix xml: <http://www.w3.org/XML/1998/namespace> .
154 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
155
156 <http://example.com/MBVL/db> a vo:Version .
157 <http://example.com/MBVL/%s> a vo:Version .\n"""%(f_name, f_name))
158 for i in range(len(false_positive)):
159 fid, j = false_positive[i]
160 clean_fid = urllib.quote(fid.split()[0])
161 if results[fid].freq > 10:
162 fp_file.write("""mbvl:db vo:absentFrom mbvl:AddChange%i .
163 next:%s a vo:Attribute .
164 mbvl:%s vo:hasAttribute next:%s .
165 mbvl:AddChange%i a vo:AddChange ;
166 vo:resultsIn next:%s .\n
167 """%(i, clean_fid, f_name, clean_fid, i, clean_fid))
```

```
168 fp_file.close()
169
170 def get_output(tax1, alg1, tax2, alg2):
171 if alg1 == 'silva_spingo':
172 t1 = [x for x in tax1 if not is_number(x)]
173 del t1[5]
174 elif alg1 == 'rdp_spingo':
175 t1 = [x for x in tax1 if not is_number(x)]
176 print t1
177 del t1[1]
178 t1[0:0] = ['Pass', 'Pass', 'Pass', 'Pass']
179 else:
180 t1 = tax1
181
182 if alg2 == 'silva_spingo':
183 t2 = [x for x in tax2 if not is_number(x)]
184 del t2[5]
185 elif alg2 == 'rdp_spingo':
186 t2 = [x for x in tax2 if not is_number(x)]
187 del t2[1]
188 t2[0:0] = ['Pass', 'Pass', 'Pass', 'Pass']
189 else:
190 t2 = tax2
191
192 if alg1.split('_')[1] != alg2.split('_')[1]:
193 for j in range(7):
194 if t1[j] == 'Pass':
195 continue
196 if j >= len(t2):
197 if t1[j] != 'AMBIGUOUS':
198 return "\t".join([ '---', rank[j], str(t1[j:])])
199 elif j == 6:
200 if t1[j] == 'AMBIGUOUS':
201 return "\t".join(['+++', rank[j], str(t2[j:])])
202 elif t1[j] != '_'.join(t2[5:7]):
203 return "\t".join([ '>>>', rank[j], t1[j], str(tax2[5:])])
```

```
204 elif t1[j] == 'AMBIGUOUS':
205 return "\t".join([ '+++', rank[j], str(t2[j:])])
206 elif t1[j] != t2[j]:
207 return "\t".join([ '>>>', rank[j], str(t1[j]), str(t2[j:])])
208 return None
209 elif alg1.split('_')[1] == 'spingo':
210 for j in range(7):
211 if t1[j] == 'Pass' or t2[j] == 'Pass':
212 continue
213 if t1[j] == 'AMBIGUOUS':
214 if t2[j] == 'AMBIGUOUS':
215 return None
216 else:
217 return "\t".join(['+++', rank[j], str(t2[j:])])
218 elif t2[j] == 'AMBIGUOUS':
219 return "\t".join(['---', rank[j], str(t1[j:])])
220 elif t1[j] != t2[j]:
221 return "\t".join(['>>>', rank[j], str(t1[j]), str(t2[j])])
222 return None
223 elif alg1.split('_')[1] == 'gast':
224 if len(t1) > len(t2):
225 return "\t".join(['---', rank[len(t2)], str(t1[len(t2):])])
226 elif len(t1) < len(t2):
227 return "\t".join(['+++', rank[len(t1)], str(t2[len(t1):])])
228 else:
229 for j in range(len(t1)):
230 if t1[j] != t2[j]:
231 return "\t".join(['>>>', rank[j], str(t1[j]), str(t2[j])])
232 return None
233 else:
234 return -1
235
236 mock = ['Acinetobacter baumannii',
237 'Actinomyces odontolyticus',
238 'Bacillus cereus',
239 'Bacteroides vulgatus',
```

```
'Clostridium beijerinckii',
'Deinococcus radiodurans',
'Enterococcus faecalis',
'Escherichia coli',
'Helicobacter pylori',
'Lactobacillus gasseri',
'Listeria monocytogenes',
'Neisseria meningitidis',
'Porphyromonas gingivalis',
'Propionibacterium acnes',
'Pseudomonas aeruginosa',
'Rhodobacter sphaeroides',
'Staphylococcus aureus',
'Staphylococcus epidermidis',
'Streptococcus agalactiae',
'Streptococcus mutans',
'Streptococcus pneumoniae']

genus = [x.split()[0] for x in mock]

family = ['Moraxellaceae',
'Actinomycetaceae',
'Bacillaceae',
'Bacteroidaceae',
'Clostridiaceae',
'Deinococcaceae',
'Enterococcaceae',
'Enterobacteriaceae',
'Helicobacteraceae',
'Lactobacillaceae',
'Listeriaceae',
'Neisseriaceae',
'Porphyromonadaceae',
'Propionibacteriaceae',
'Pseudomonadaceae',
'Rhodobacteraceae',
```

```
276 'Staphylococcaceae',
277 'Staphylococcaceae',
278 'Streptococcaceae',
279 'Streptococcaceae',
280 'Streptococcaceae']
281
282 rank = ['Kingdom', 'Phylum', 'Class', 'Order', 'Family', 'Genus',
    ↪  'Species', 'Strain']
283
284 if __name__ == "__main__":
285 fList = ['silva_spingo', 'silva_gast', 'rdp_spingo', 'rdp_gast']
286 #f1_name = fList[1]
287 #f2_name = fList[3]
288 #f1 = file_parse(f1_name)
289 #f2 = file_parse(f2_name)
290
291 for i in fList:
292 f1 = file_parse(i)
293 #out = file('silva_rdp_gast.txt', 'w')
294 #for i in sorted(f1.keys()):
295 #        x = get_output(f1[i].tax, f1_name, f2[i].tax, f2_name)
296 #        if not x == None:
297 #                out.write("\t".join([i,x])+"\n")
298 #out.close()
```

APPENDIX G
COLD LAND PROCESS FIELD EXPERIMENT CHANGE LOG GENERATOR

```
from glob import glob
from datetime import datetime
import csv
import dbfread
import sys

class Version:
unique_id = None

def __eq__(self, other):
if isinstance(other, self.__class__):
return self.unique_id == other.unique_id
else:
return self.unique_id == other

class StratVer(Version):

def __init__(self, headers, values):
v = [i.strip() for i in csv.reader([values]).next()]
self.data = {}
self.load_strat(headers, v)
self.unique_id = (self.data['PIT_NAME'], self.data['IOP'],
 ↪  int(self.data['TOP']), int(self.data['BOT']))

def load_strat(self, headers, v):
for i in range(len(headers)):
if i == 2:
self.data[headers[i]] = datetime.strptime(v[i], "%Y-%m-%d").date()
elif i > 2 and i < 14:
if '.' in v[i]:
```

```
self.data[headers[i]] = float(v[i])
else:
self.data[headers[i]] = int(v[i])
else:
self.data[headers[i]] = v[i]

class SummVer(Version):

def __init__(self, headers, values):
v = [i.strip() for i in csv.reader([values]).next()]
self.data = {}
self.load_summ(headers, v)
self.unique_id = (self.data['PIT'], self.data['IOP'])

def load_summ(self, headers, v):
for i in range(len(headers)):
if i == 3:
try:
self.data[headers[i]] = datetime.strptime(v[i], "%Y-%m-%d").date()
except ValueError:
self.data[headers[i]] = None
elif (i > 3 and i < 7) or (i > 8 and i < 11) or (i > 11 and i < 34):
try:
self.data[headers[i]] = int(v[i])
except ValueError:
pass
try:
self.data[headers[i]] = float(v[i])
except ValueError:
self.data[headers[i]] = v[i]
else:
self.data[headers[i]] = v[i]

def helper_factory( section, mode, arg1=None):
if mode == "strat":
if section == "names":
```

```
66  return ("/data/ice/shape_files/pit_iop_v2_strat.dbf", "strat")
67  elif section == "key":
68  return "Key = (PIT, IOP, TOP, BOT)\n\n"
69  elif section == "mapper":
70  return strat_col_map
71  elif section == "table_ids":
72  return [(r['PIT'], r['IOP'], r['TOP'], r['BOT']) for r in arg1]
73  elif mode == "summary":
74  if section == "names":
75  return ("/data/ice/shape_files/pit_iop_v2_summary.dbf", "summary")
76  elif section == "key":
77  return "Key = (PIT, IOP)\n\n"
78  elif section == "mapper":
79  return summ_col_map
80  elif section == "table_ids":
81  return [(r['PIT'], r['IOP']) for r in arg1]
82
83  def strat_col_map(index, mode):
84  if mode == "ascii":
85  if index == "DATE_":
86  return "DATE"
87  elif index == "PIT":
88  return "PIT_NAME"
89  elif mode == "shape":
90  if index == "DATE":
91  return "DATE_"
92  elif index == "PIT_NAME":
93  return "PIT"
94  return index
95
96  def summ_col_map(index, mode):
97  if mode == "ascii" and index == "DATE_":
98  return "DATE"
99  elif mode == "shape" and index == "DATE":
100 return "DATE_"
101 return index
```

```
102
103 def ice_import(category):
104 if category == "strat":
105 fn = glob("/data/ice/ascii/*strat.csv")
106 elif category == "summary":
107 fn = glob("/data/ice/ascii/*summary.csv")
108
109 entries = {}
110 for i in fn:
111 f = open(i)
112 ll = f.readlines()
113 headers = [c.strip() for c in csv.reader([ll[0]]).next()]
114 for j in ll[2:]:
115 if category == "strat":
116 sv = StratVer(headers, j)
117 elif category == "summary":
118 sv = SummVer(headers, j)
119 entries[sv.unique_id] = sv
120 f.close()
121 return headers, entries
122
123 def generate_logs(table_name, text_name, f):
124 table = dbfread.DBF(table_name)
125 table_ids = helper_factory("table_ids", text_name, table)
126
127 headers, text_entries = ice_import(text_name)
128
129 f.write("Shape Entries not in Ascii Files\n")
130 for i in table_ids:
131 if i not in text_entries.keys():
132 f.write(str(i)+"\n")
133 f.write("\n")
134
135 f.write("Shape Columns not in Ascii Files\n")
136 for i in table.field_names:
137 if i not in [column_map(j, "shape") for j in headers]:
```

```
138 f.write(i+"\n")
139 f.write("\n")
140
141 f.write("Ascii Entries not in Shape Files\n")
142 for i in sorted(text_entries.keys()):
143 if i not in table_ids:
144 f.write(str(i)+"\n")
145 f.write("\n")
146
147 f.write("Ascii Columns not in Shape Files\n")
148 for i in headers:
149 if i not in [column_map(j, "ascii") for j in table.field_names]:
150 f.write(i+"\n")
151 f.write("\n")
152
153 c = 0
154 f.write("Modified Entries\n")
155 for r in table:
156 if text_name == "strat":
157 i = (r['PIT'], r['IOP'], r['TOP'], r['BOT'])
158 elif text_name == "summary":
159 i = (r['PIT'], r['IOP'])
160
161 if i in text_entries.keys():
162 v = text_entries[i]
163 start = True
164 for j in table.field_names:
165 if column_map(j, "ascii") not in v.data.keys():
166 pass
167 elif r[j] == 'NoData' and v.data[column_map(j, "ascii")] == '-999':
168 pass
169 elif r[j] != v.data[column_map(j, "ascii")]:
170 if start:
171 f.write("%s\n"%(str(v.unique_id)))
172 f.write("Column|\tShape|\tAscii\n")
173 start = False
```

```
174 f.write("%s|\t%s|\t%s\n"%(j, r[j], v.data[column_map(j, "ascii")]))
175 #print v.unique_id, j, r[j], v.data[attribute_map(j)]
176 c += 1
177 if not start:
178 f.write("\n")
179
180 if __name__ == "__main__":
181 if "-strat" in sys.argv:
182 table_name, text_name = helper_factory("names", "strat")
183 elif "-summ" in sys.argv:
184 table_name, text_name = helper_factory("names", "summary")
185
186 change_log = "changelog_"+text_name+".txt"
187 f = open(change_log, 'w')
188 f.write("Change Log %s to ascii files\n"%(table_name.split('/')[-1]))
189 f.write(helper_factory("key", text_name))
190
191 column_map = helper_factory("mapper", text_name)
192 generate_logs(table_name, text_name, f)
193 f.close()
```

G.1 Versions as a Class

In the CLPX data, versions were encapsulated into individual classes so that computations between versions could be standardized and automated. The code could handle ingesting a variety of different formats of data while keeping the core comparison functions independent of individualized fields.

APPENDIX H
EARTH OBSERVING LABORATORY ANALYSER

```
import csv
from datetime import datetime

class dataset:
def __init__(self, d_id, d_title):
self.versions = {}
self.num = int(d_id)
self.title = d_title

def add_file(self, v_num, v_pub, v_crt, v_mod, f_name, f_crt, f_rev,
↪  f_notes):
if v_num not in self.versions:
self.versions[v_num] = eol_ver(v_num, v_pub, v_crt, v_mod)
self.versions[v_num].add_file(f_name, f_crt, f_rev, f_notes)

def __repr__(self):
out = ["%i: %s"%(self.num, self.title)]
for i in sorted(self.versions.keys()):
out.append(self.versions[i].string(4))
return "\n".join(out)

class eol_ver:
def __init__(self, v_num, t1, t2, t3):
self.num = v_num
self.files = []
self.v_pub = datetime.strptime(t1, "%Y-%m-%d %H:%M:%S")
self.v_crt = datetime.strptime(t2, "%Y-%m-%d %H:%M:%S")
self.v_mod = datetime.strptime(t3, "%Y-%m-%d %H:%M:%S")

def add_file(self, f_name, t1, t2, notes):
new_f = eol_file(f_name, t1, t2, notes)
```

```
31 self.files.append(new_f)
32
33 def string(self, indent):
34 ind = ' '*indent
35 out = [ind+"%s: %s    %s    %s"%(self.num, str(self.v_pub),
    ↪  str(self.v_crt), str(self.v_mod))]
36 for i in self.files:
37 out.append(i.string(indent+4))
38 return "\n".join(out)
39
40 def __repr__(self):
41 out = ["%s: %s    %s    %s"%(self.num, str(self.v_pub),
    ↪  str(self.v_crt), str(self.v_mod))]
42 for i in self.files:
43 out.append("    "+i.__repr__())
44 return "\n".join(out)
45
46 class eol_file:
47 def __init__(self, f_name, t1, t2, notes):
48 self.name = f_name
49 self.f_create = datetime.strptime(t1, "%Y-%m-%d %H:%M:%S")
50 self.f_revise = datetime.strptime(t2, "%Y-%m-%d %H:%M:%S")
51 self.f_notes  = notes
52
53 def string(self, indent):
54 out = ' '*indent
55 return out+self.name
56
57 def __repr__(self):
58 return self.name
59
60 def import_eol(fname):
61 f = open(fname)
62 f_reader = csv.reader(f, delimiter='|')
63 f_reader.next()
64 header = [i.strip() for i in f_reader.next()[1:-1]]
```

```
65 f_reader.next()
66
67 datasets = {}
68 for i in f_reader:
69 row = [j.strip() for j in i[1:-1]]
70 if len(row) < 10:
71 continue
72 if int(row[0]) not in datasets:
73 datasets[int(row[0])] = dataset(int(row[0]), row[1])
74
75 datasets[int(row[0])].add_file(row[2], row[3], row[4], row[5],
   ↪ row[6], row[7], row[8], row[9])
76 return datasets
77
78 if __name__ == "__main__":
79 filedir  = "/data/EOL/"
80 #filename = "dataset_files_version_metadata.txt"
81 filename = "EOL_dataset_version_metadata.txt"
82
83 datasets = import_eol(filedir+filename)
84 print len(datasets.keys())
85
86 num_ver = {}
87 for i in datasets.keys():
88 how_many_versions = len(datasets[i].versions.keys())
89 if how_many_versions not in num_ver:
90 num_ver[how_many_versions] = 1
91 else:
92 num_ver[how_many_versions] += 1
93 for i in sorted(num_ver.keys()):
94 print i, num_ver[i]
```