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

ACKNOWLEDGMENT

ABSTRACT

Data sets invariably require versioning systems to manage changes due to an imperfect collection environment. While importance grows, 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 PAV. The model captures addition, invalidation, and modification relationships between versions to provide change log-like characterization of the differences. This approach demonstrated increased expressibility of change interactions, but encountered issues with space scalability. The model's generation also revealed a four step process to conduct versioning: validation, mapping, computation, and publishing. Quantifying these changes also provided a numerical basis for evaluating the GCMD Keywords taxonomy's adopted identification scheme. It also demonstrates the ability of versioning methods to actively influence scientific designs through performance assessment.

CHAPTER 1
INTRODUCTION

1.1 Introduction

> 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. This basic identification method has given rise to a plethora of versioning systems used widely across a landscape of software and data. They help scientific workflows avoid losing work by managing transitions and changes while in operation [2]. They provide necessary documentation which informs the transition to new methods and procedures [3]. They provide accountability for the value of a project's data set when considering an agency's continued funding [4]. The natural evolution of these 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.

1.2 Definitions of Version

Using 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

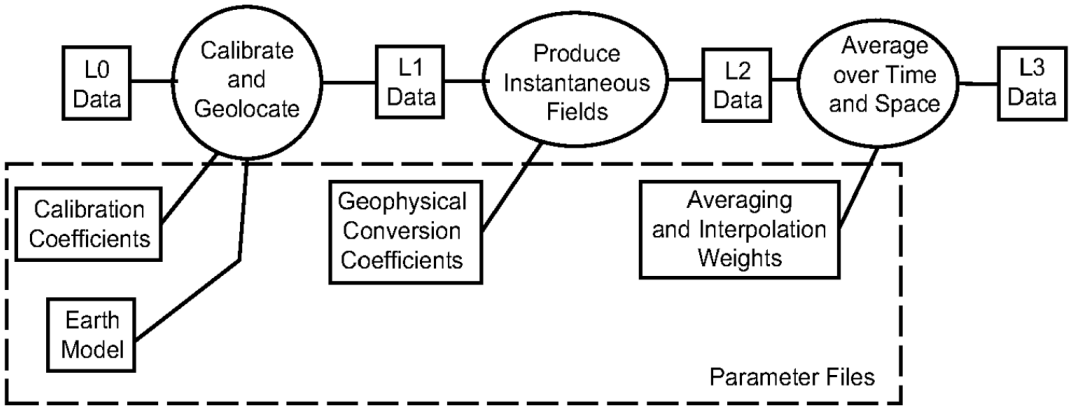


Figure 1.1: NASA organizes its data into three levels depending on the amount of aggregation and the distance the data is removed from the original sensor measurements. Figure 1 from [1]

properties. In order to determine when these properties have changed, he leverages the NASA 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 1.5.2, 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 uni-

formly 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 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.

1.3 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 [8] [9]. 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 [10]. 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 [11]. Many applications have adopted a tree-like structure which is further propagated by software versioning managers (SVM) [12]. The advantage being well established graph theory methods can be applied to complex objects relationships in complex environemtns [13]. The growing population of web documents, however, presents a new smrgsbord of complicated data which will need scalable solutions [14].

1.3.1 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

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

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

Figure 1.2: Table of predominant identifiers used in science. From Duerr, et al. [16]

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 [15]. 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 [16]. 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 1.2, DOIs represent the most suitable identifier used for citation in scholarly literature [16]. 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 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 [17]. 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 [18].

As digital methods have evolved, so have digital libraries. The documents that digital libraries store are no longer constrained by physical organization [19]. 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 [20]. The regularizing edition tracking methods significantly improved the response time and relevancy of the library services.

1.3.2 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 [21]. 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 1.3. Not long after, the Concurrent Versions System (CVS) gained popularity with methods allowing multiple users to concurrently develop code to a central repository [22]. The most popular modern SVM is GIT which also allows concurrent development but enables distributed repositories [23]. 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

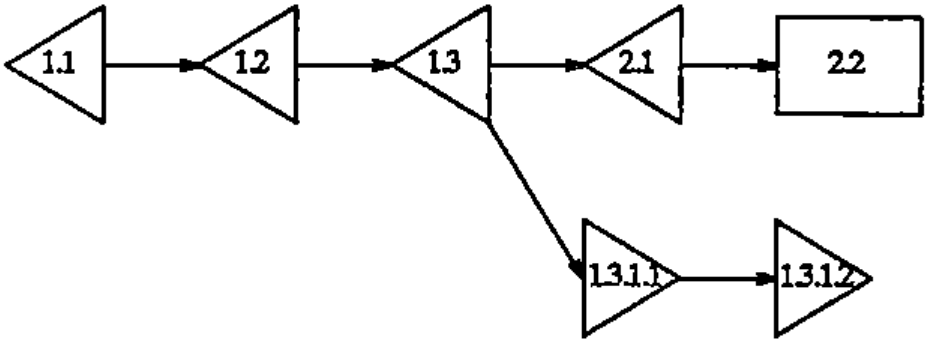


Figure 1.3: Commit history of an object in RCS with changes in the main line stored as back deltas and side branches stored as forward deltas. Figure 5 in [21]

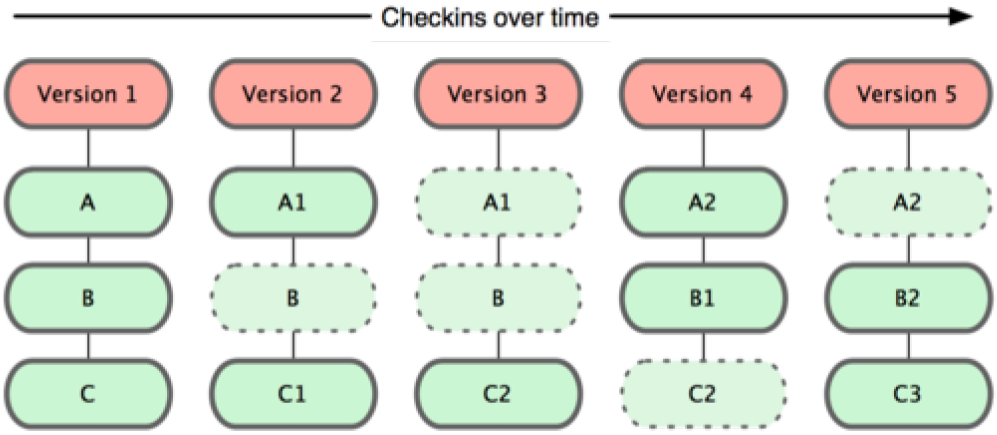


Figure 1.4: GIT stores changes in the repository as snapshots of individual files. Figure 1.5 from [23]

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 1.4 demonstrates an example of how GIT employs storage space for multiple versions [23]. 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 [24].

1.3.3 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 into related groupings [25]. 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 [26]. 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 [27]. Proell and Rauber have investigated tracking data queries instead of the database as a more scalable solution to reproduce data [28]. The queries can then be used as publication citations to provide scalable, reproducible references to older data [29] [30].

1.3.4 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 [31]. 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 [32]. 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 [17]. 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 [33].

1.3.5 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 [34]. 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 [35]. Klein and Fensel have found that when the changes are discovered, both forward and backward compatibility must be established for clear ontology versioning [36]. 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 [37] [38]. The method heavily relies on the context of stored data.

1.4 Version Models

Version models provide a visual a 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[39]. Adding files already in the set created a new 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 [40]. Their model, shown in Figure 1.5, 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.

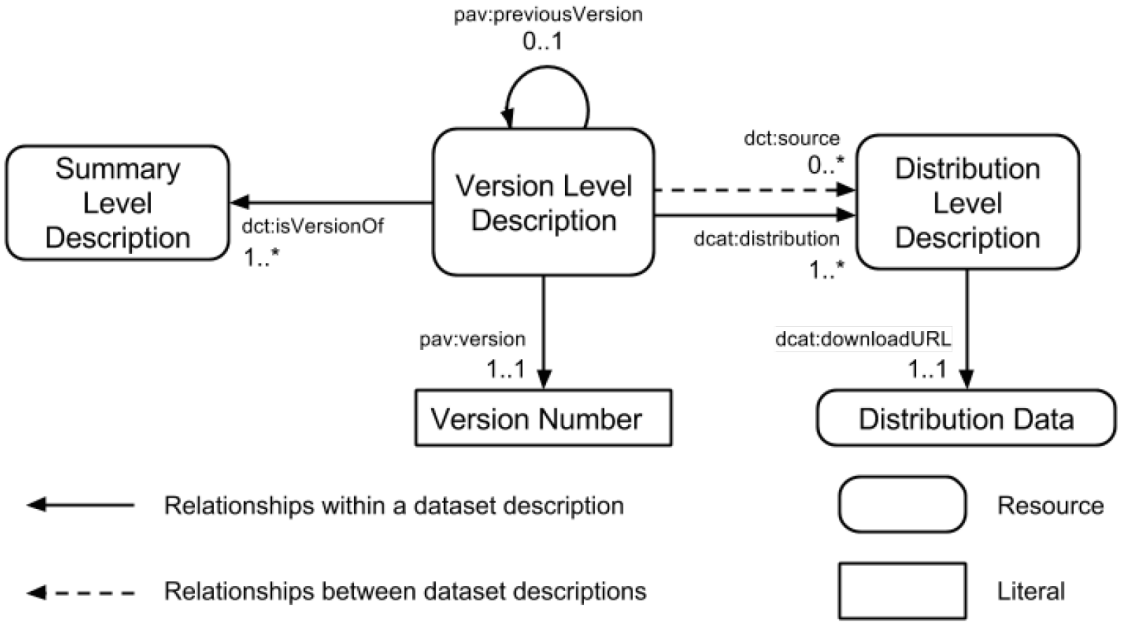


Figure 1.5: Data model from the Health Care and Life Sciences Interest Group separating data into three levels: works, versions, and instances. From Dummontier, et al. [40]

From his definition of versions, Barkstrom also outlines an hierarchical version model as seen in Figure 1.6. The model features additional intermediary levels than the HCLS's model, following NASA's data curation practices [41]. Each edge in the tree signifies a difference with other objects at the same depth, but the model does not provide a mechanism to explain the difference.

1.5 Provenance Ontologies

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 [42]. The technique provides a powerful way to express and imply sequences of relationships between different versions and characterize the manner of their relation.

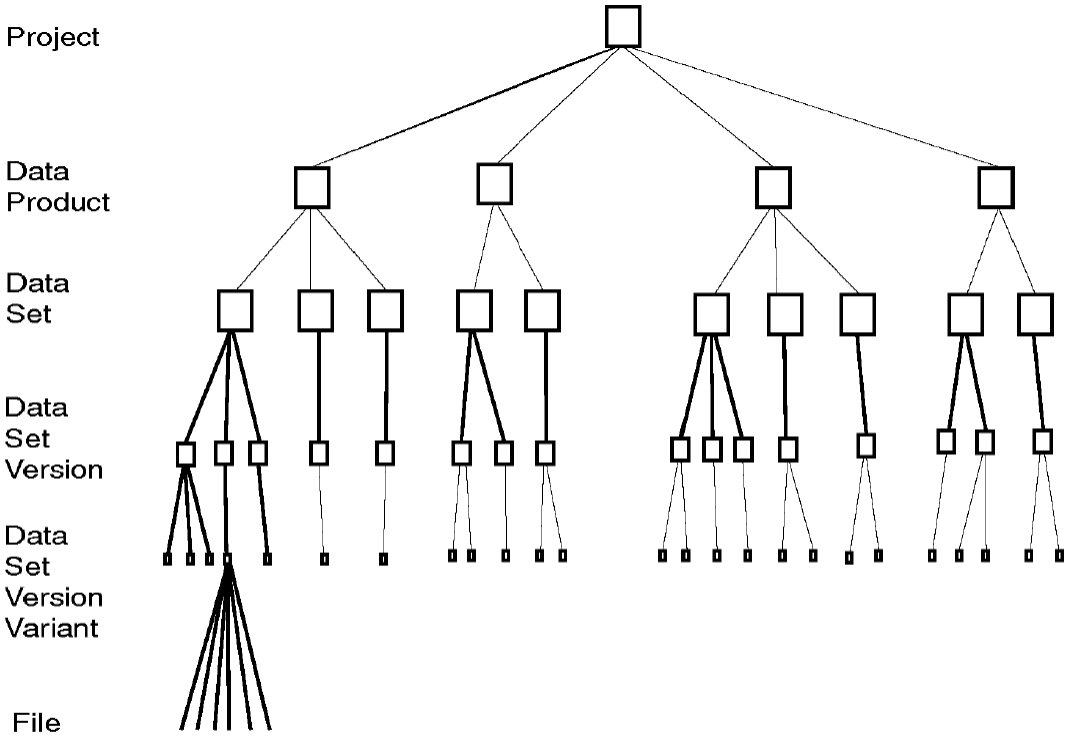


Fig. 2. Hierarchical groupings of files from data products to individual files

Figure 1.6: Visual representation of grouping hierarchy. From [1]

1.5.1 Open Provenance Model

A number of linked data models include versioning concepts such as the Open Provenance Model (OPM) [43]. 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 [44]. 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 [45]. 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 encom-

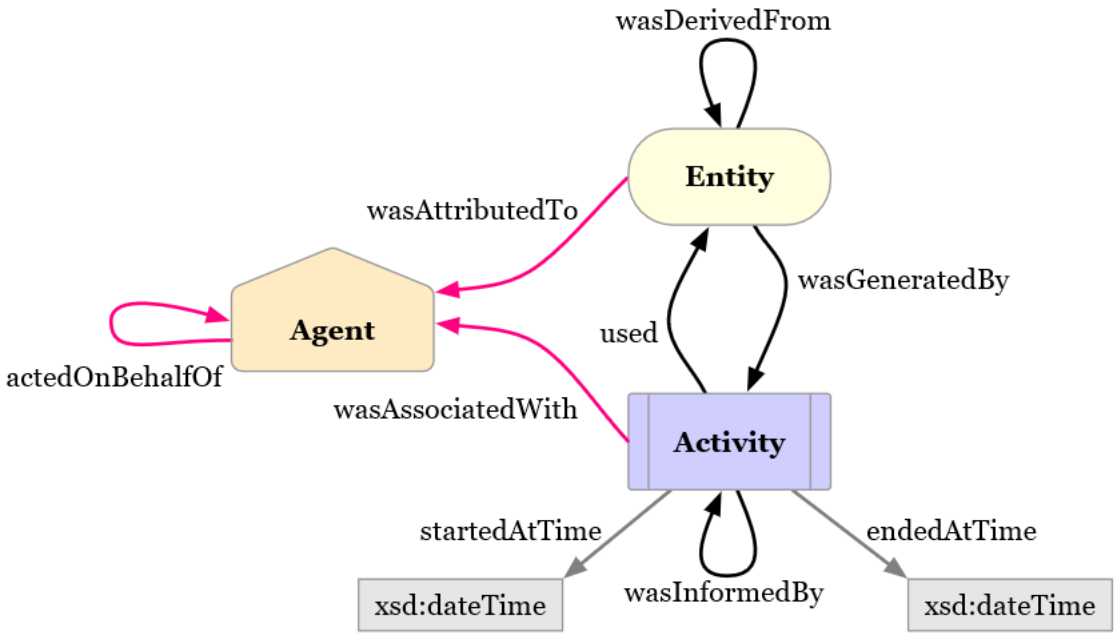


Figure 1.7: Diagram of the PROV Ontology. Figure 1 from [6]

passes the engagement which provenance models view versions, without further need to explore the derivation's content.

1.5.2 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 1.7 [46] [47]. The recommendation uses a conceptual model relating activities, agents, and entities to describe data production lineage [48] [49] [50]. 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 [51]. The conceptual model's expression occurs through the PROV Ontology (PROV-O), which can be conveyed through various resource description languages [52] [53]. The ontology is further formalized into a functional notation for easier human consumption [54] [55]. 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 [56] [57].

PROV has provided a major contribution in maintaining the quality and re-

producibility of data sets and reporting in the National Climate Assessment (NCA) [58]. 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 [59]. 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 [60]. 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. [61].

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.

1.5.3 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" [62]. 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 [62]. 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" [63]. 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.

1.5.4 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 [64]. 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" [65]. 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" [66]. 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," [67]. The concept aligns well with other versioning systems, although deletion may be a strong assertion.

1.6 Change Logs

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 [68]. 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 provide 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 these logs can supply 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. 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 [69]. 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 [70]. 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 [71]. 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.

1.7 Introduction of Use Cases

1.7.1 Use Case 1: Linked Data Change Log

Needed a project with at least two versions, and the objects had to have content differences. The single project constraint makes sure they're from the same work. Excel spreadsheets allow simultaneous instances of each version and well supported tools for access.

1.7.2 Use Case 2: Multi-version Change Distance

1.8 Hypothesis Statement

Large data collection endeavors necessitate the development and deployment of versioning systems to manage change propagating through their data. Advancing beyond identifier comparisons and delving into capturing substantive change can significantly improve the ability to standardize version communications and transition. The work in the following chapters will contribute to expanding change capture by defining a more expressive linked data versioning model. The model will allow current versioning documentation to become machine-consumable. A better method will be developed to provide a quantitative basis for the assignment of version identifiers.

1.9 Contributions

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 [72]. In comparing their methods, many systems also share the implementation of common versioning operations, suggesting an avenue for fundamental versioning properties. While SVMs prefer to adopt th dot-decimal identifier, DOIs 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 Data Versioning Operations

Among all the systems surveyed in Section 1.3, 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 [73]. 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 [8]. 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 [23]. 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 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 [21]. Branching mostly provides an organizational role in development by allowing developers to experiment without contaminating a stable software release. Figure 2.1 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 [13].

2.2.1 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

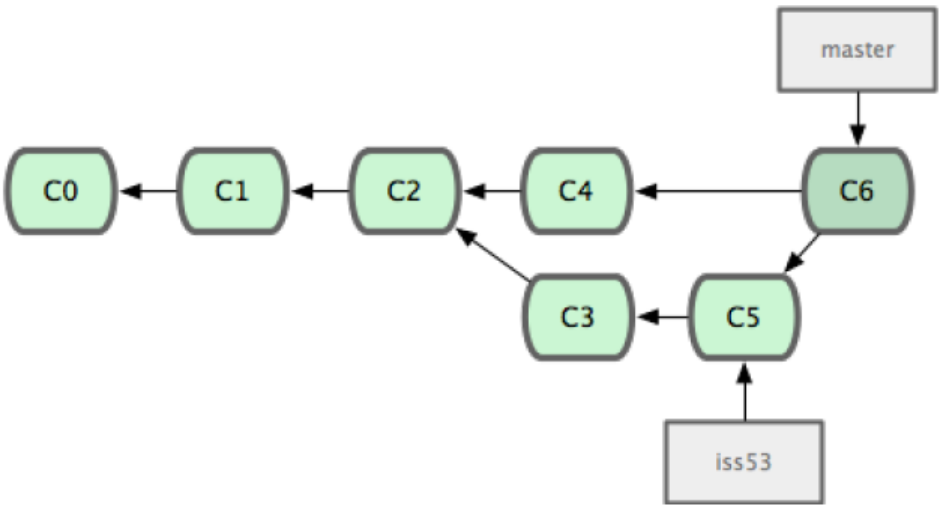


Figure 2.1: Example of a commit history with branching stored in GIT. Figure 3.17 from [23]

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.

2.3 Identifiers

The most widely identifier scheme associated with versioning is the dot-decimal identifier [12]. Whenever, a new version is made, it receives an identifier with one of the numbers incremented as seen in Figure 1.3. Such a procedure fails to communicate the extent of a change because, regardless of the amount, the identifier will increment 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 [13]. 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 [74]. A common method used to address the distinction between versions is a human-readable change log, further discussed in Section 1.6.

The discourse on DOIs highlights the importance of understanding the limitations of particular identifier schemes. With respect to Figure 1.2, no identification scheme fits the description of a scientific identifier. Duerr, et al., define a use case to make the argument that scientifically unique identifiers are necessary, “to be able

to tell that two data instances contain the same information even if the formats are different" [16]. 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 [22]. 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 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 2.2 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.

2.4 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. [75]. Figure 2.3 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

Figure 5.3: Benevolent dictator workflow

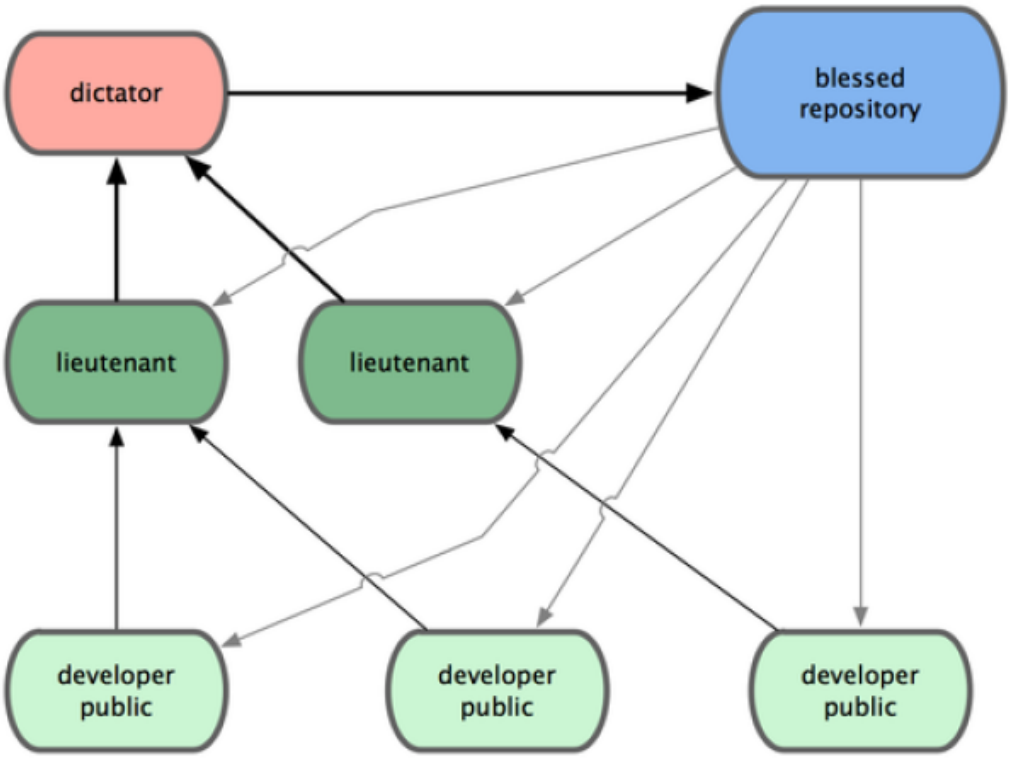


Figure 2.2: A distributed workflow to control for volatile versioning behavior. From [22].

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 [77]. The design of RDFa focuses on describing the web page's content through markup [76]. The underlying or resulting versioning data model may not conform with the format of content presented in the change log. Poor

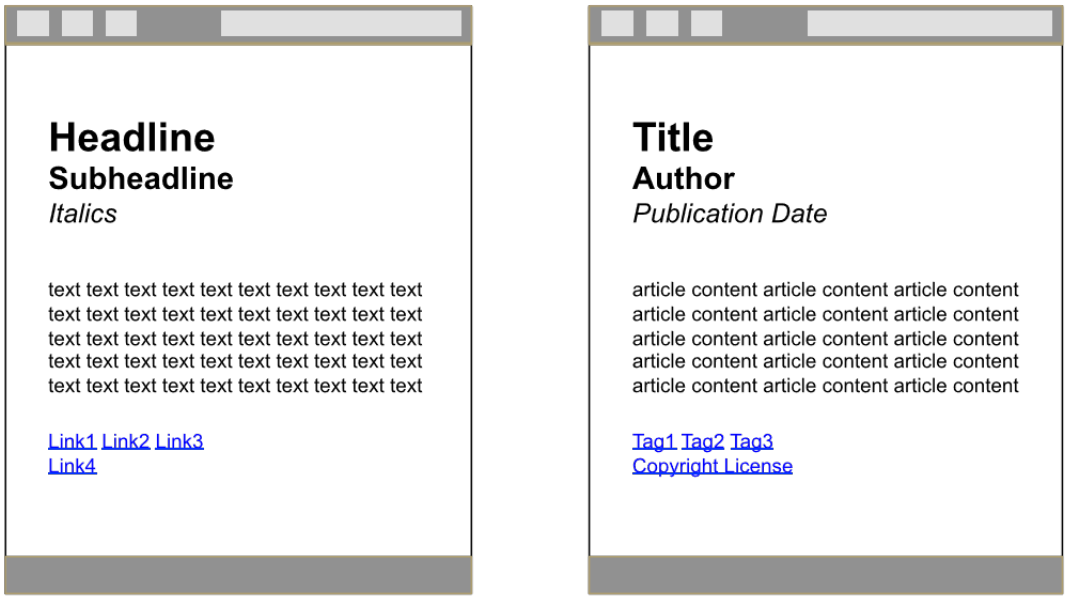


Figure 2.3: Illustration of the difference in what autonomous systems see when crawling a web page and what humans see when reading the same material. Figure 1 from [76]

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 [78]. 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.

2.5 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 [79]. Additionally, changing versions are often used to signal other applications downstream that a new version may be necessary to adopt data improvements [80]. Many efforts currently to compute a distance measure relies on data provenance. Formalizing operations on provenance remains an active field of research [81]. Other approaches relate to determining semantic similarity in trying to summarize the data set and computing a distance measure [82].

2.5.1 Provenance Distance

Previous endeavors to extract insight into data set performance or behavior have provided exciting results [83]. 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 2.5.1. 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 [84]. Node labeling simplifies the similarity measurement process by providing nodes which must match together, and greatly reduces the

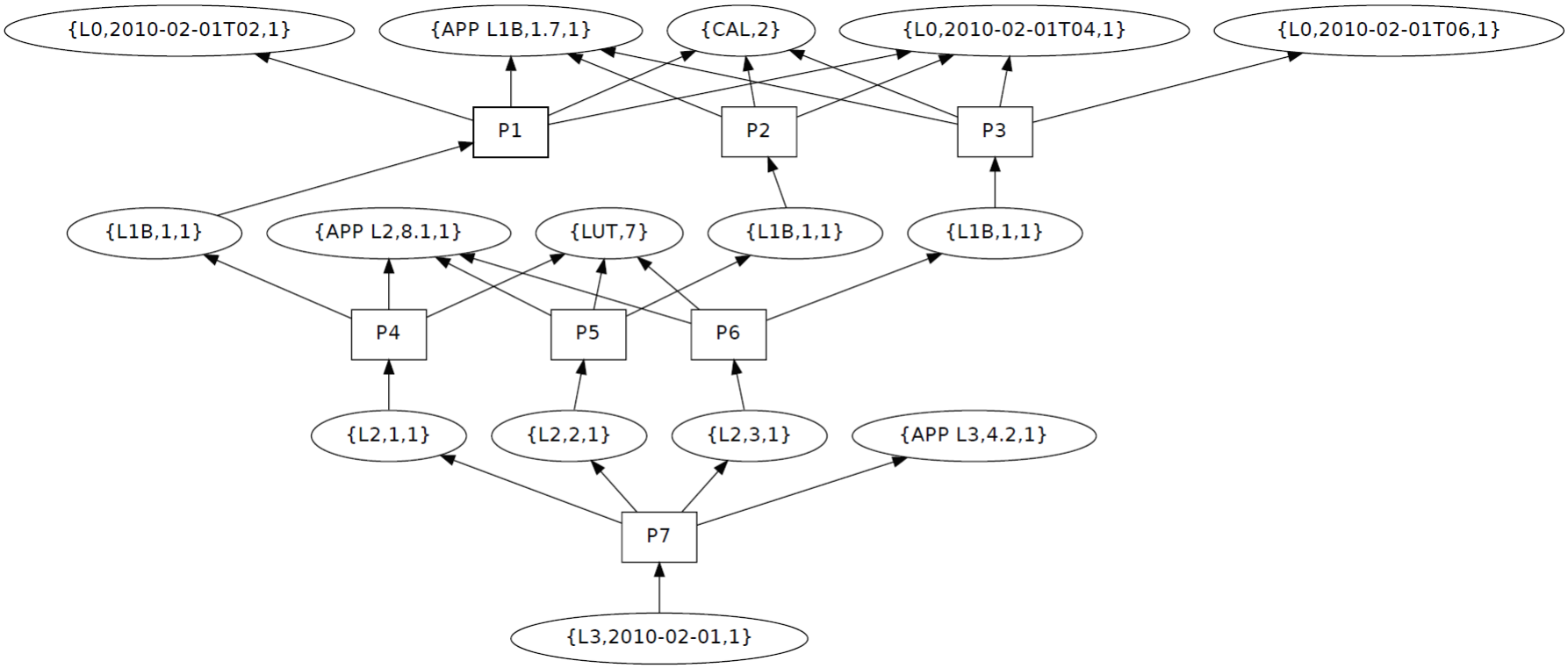


Figure 2.4: Provenance graph of a Level 3 data product, showing the inter-relations between different data products in generating the final product. Figure 2 from [80]

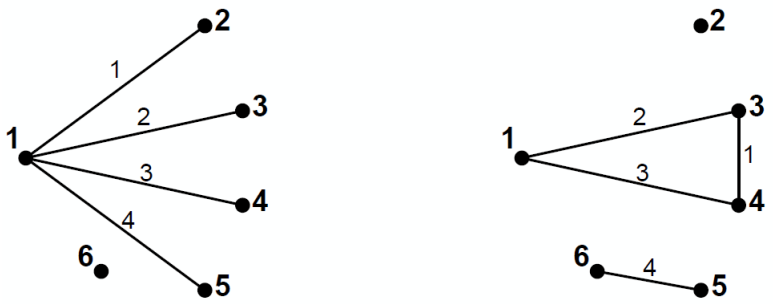


Figure 2.5: The labeled graph on the left transforms into the right graph under two edge edits. Figure 2 from [86]

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 [85]. Some variations count edge changes [86].

In Figure 2.5, 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 have been affected by the change in the calibration.

[80]. 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 [87]. These procedures focus primarily on quick retrieval and efficient storage instead of capitalizing on the latent information accessed by reasoning across data set versions [88]. 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 compa-

rable to a change log requires a more detailed understanding and description than provenance can provide [89]. 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.

2.6 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 presents a model to create a versioning graph.

CHAPTER 3
MODEL SPECIFICATION

3.1 Introduction

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 1.5, and Barkstrom model, Figure 1.6, 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 1.6, the left-most leaf node and the right-most leaf node differ by three changes 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 rely on structure to imply. The 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.

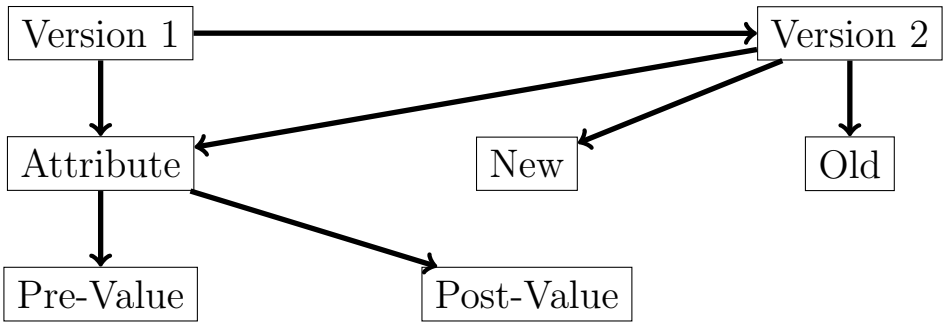


Figure 3.1: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

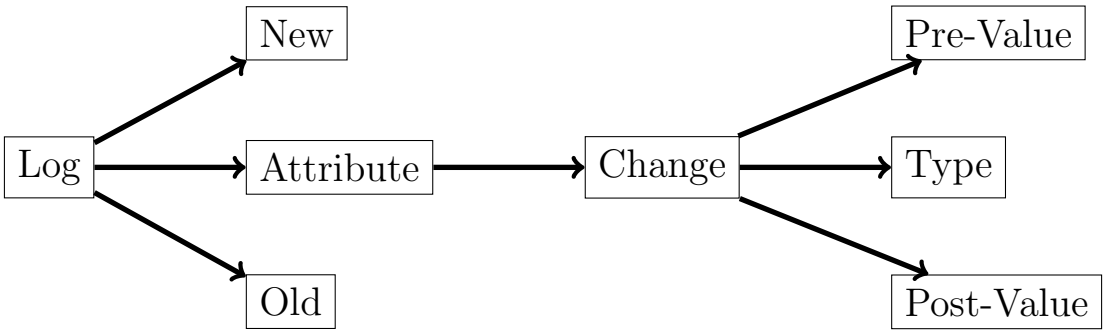


Figure 3.2: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

3.2 Initial Approaches

3.3 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

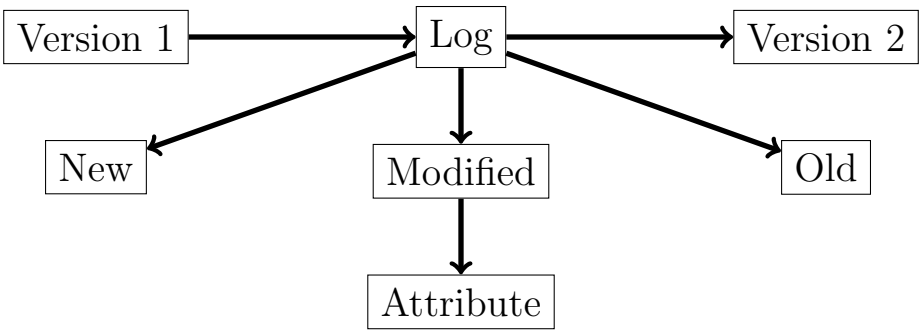


Figure 3.3: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

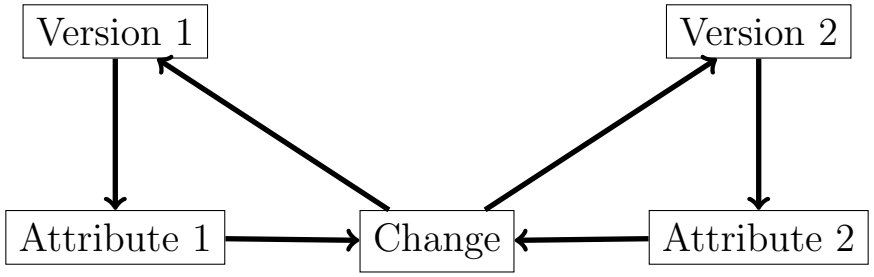


Figure 3.4: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

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*.

3.3.1 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

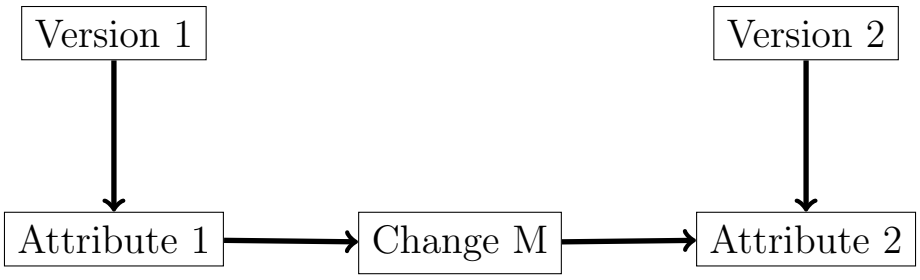


Figure 3.5: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

this distinction, objects will be referred to as the left-hand **version** or left-hand **attribute** when they are not sequentially or temporally related.

3.4 Model Changes

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.

3.4.1 Modification

The **modification** relation occurs when an **attribute** appears in both **versions** and their values are different. In Figure 3.5, 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.

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

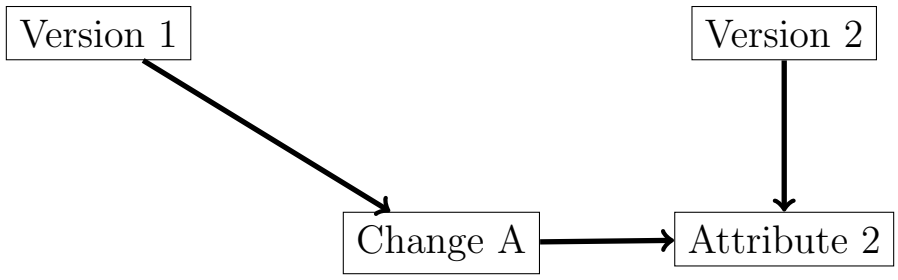


Figure 3.6: Model of the relationships between Versions 1 and 2 when adding an Attribute 2 to Version 2 as a result of Change A

outside the scope for this project. In addition, the model would essentially begin storing a copy of the data set, leading to space and redundancy concerns.

3.4.2 Addition

In Figure 3.6, 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". 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 may be true. The construction was also chosen to create a symmetric orientation with the **invalidation** change.

3.4.3 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 3.7, the invalidation change concept matches to the Version 2 object. Just like in **addition** model, this construction maintains a link between the two **version** objects. In this case, it makes more conceptual sense, however, because "Version 2" invalidates "Attribute 1" by omitting it.

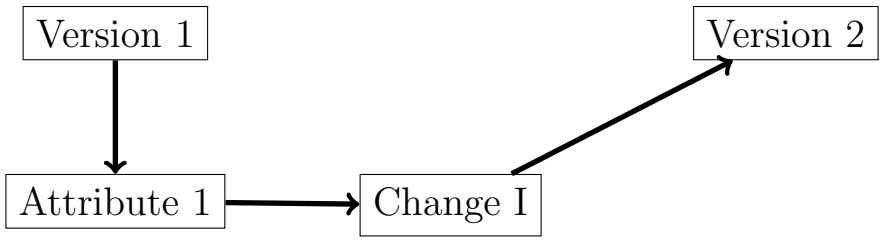


Figure 3.7: Model of the relationships between Versions 1 and 2 when invalidating Attribute 1 from Version 1 as a result of Change I

3.5 Summary

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 together **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.

CHAPTER 4
RESULTS

4.1 Introduction

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 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 Utilized Data Sets

4.2.1 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 8 files with 194 columns each. The next version of the database, published March 8, 2015, compiles all the data into a single file and 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.

4.2.2 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 Chapter 3. The data from this work is also more processing friendly, making it agreeable to automatic change log generation.

4.2.3 GCMD Keywords

The Global Change Master Directory (GCMD) is a metadata repository used by NASA to store records of its available data sets [92]. They employ a set of keywords to make NASA Earth Science data sets searchable. These words tag and label datasets into strictly defined categories [93]. GCMD Keywords do not qualify as a standard web ontology since it does not constitute a class hierarchy. The management team stored early versions of the keywords in Excel spreadsheets, and a centralized distribution system was not used until 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, were 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 [94]. 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 [95].

Table 4.1: List of versions available in the KMS.

June 12, 2012	7.0	8.0	8.1	8.2	8.3	8.4	8.4.1	8.5

Table 4.2: 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

4.2.4 MBVL 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 [96]. 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 4.2. The original population's census is not available to the author, and only the list of species are known, forming the first data set in this section. These sequences are then grouped and classified by a 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 difference between the data sets are the ranks assigned to each sequence.

4.3 Implementing the Versioning Model

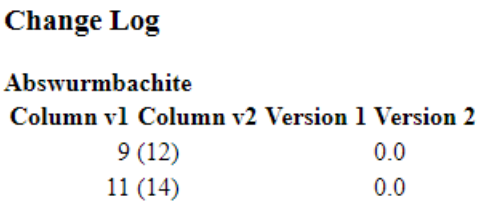
The following subsections detail the steps used to implement a versioning graph using the model defined in Chapter 3 and the challenges encountered. Section 4.3.1

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.

4.3.1 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. 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



Change Log

Abswurmbachite

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

Figure 4.1: Abswurmbachite entry in the Copper Dataset Change Log

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.

4.3.2 Produce Change Log

Change logs were generated for both the Noble Gas data set and the Copper data set. Following the practices of other change logs, the documents present before and after values for comparison which can be seen in Figure 4.1. The partial snapshot of the change log shows that column numbers in the Copper data set also accompany the values to identify the data's location in the spreadsheet. Very little natural language is used in this 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 4.1 shows the text necessary to layout the first four lines of Figure 4.1. While the content only shows four lines, the underlying markup takes up three and a half times as many lines. Line 2 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 an un-logical 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 4.1: Abswurmbachite RDFa

After encountering the limitations of using RDFa to include the versioning graph into the change log, JSON-LD was used. The new format does not rely on the structure of visible content to determine the syntax triples use to be included in the change log. Listing 4.2 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.

```
<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#ModifyChangeAbswurmbachite12"
},
{
       "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.
          jsonld",
```

```
24        "@id": "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/
             CUjsonlog.html#ModifyChangeAbswurmbachite12",
25        "@type": "vo:ModifyChange",
26        "resultsIn": "http://CUdb.com/v2/AttributeAbswurmbachite12"
27 },
28 {
29        "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.
             jsonld",
30        "@id": "http://CUdb.com/v2/AttributeAbswurmbachite12",
31        "@reverse": {
32               "hasAttribute": "Version2"
33        },
34        "@type": "vo:Attribute",
35        "label": "Primary"
36 }
37 ]
38      </script>
39    </tr>
40  </table></div><br>
```

Listing 4.2: Abswurmbachite JSON-LD

The encoded change logs struggle with intense document size. The Copper Minerals data set encoded in RDFa is 1.7 MB and JSON-LD is 3.3 MB. In the Noble Gas data set, the RDFa and JSON-LD change log sizes are 59 and 60 MB, respectively. The Noble Gas change logs often do not load in a browser.

Table 4.3: Sizes of change log encodings.

Dataset	No Encoding (MB)	RDFa (MB)	JSON-LD (MB)
Copper Minerals	0.137	1.7	3.4
Noble Gas	5.4	59	124

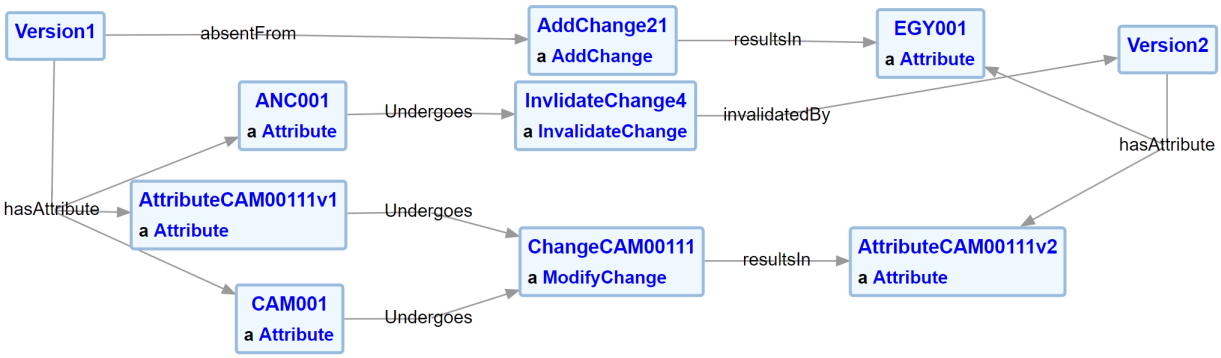


Figure 4.2: Some initial entries from versions 1 and 2 of the Noble Gas data set

4.3.3 Generate Versioning Graph

The versioning graphs presented in this section were created by extracting triples from the associated change log. 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 4.2 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 4.3. 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

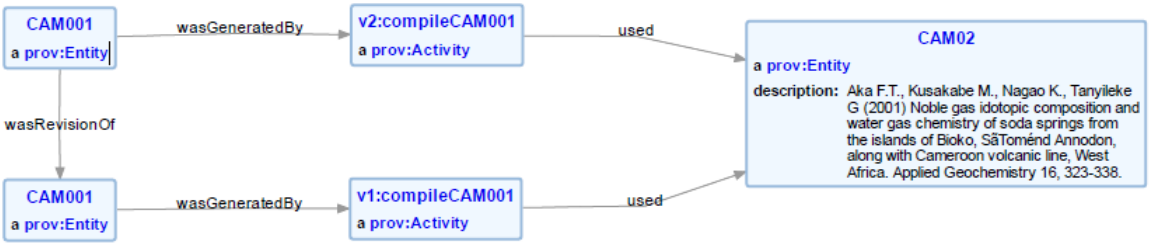


Figure 4.3: Provenance graph for the CAM001 entry of the Noble Gas Database. Other than the labels, the structure of each data object is very much the same.

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 4.2 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.

Listing 4.3 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 4.2. 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 4.3: Noble Gas Add in Turtle

Figure 4.3.3 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.

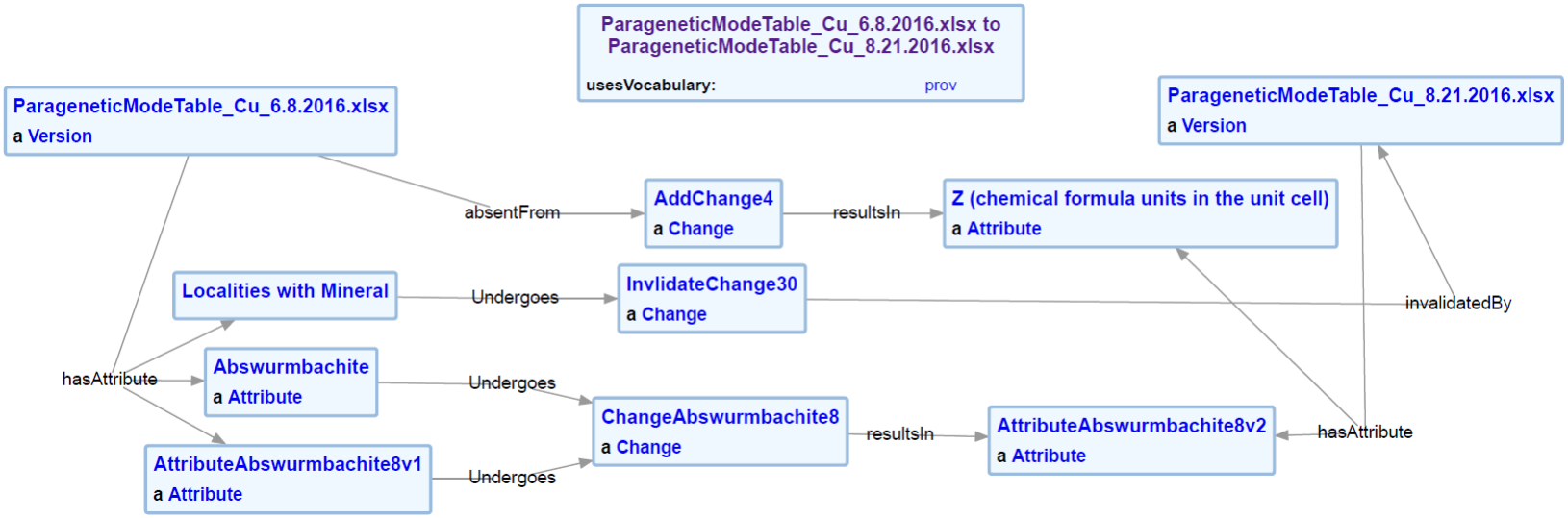


Figure 4.4: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications.

4.3.4 Graphs with Multiple Versions

Figures 4.2 and 4.3.3 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 4.3.4 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 4.2 and Figure 4.3.4 refer to columns. Figure 4.2 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 4.3.4 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 identifier changes in the same column across multiple rows.

4.4 GCMD

4.4.1 GCMD Versioning Graph

The Global Change Master Directory establishes the context that each **manifestation** of their keyword list are related versions. Since the unique identifier for each keyword remains the same across versions, they 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. 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

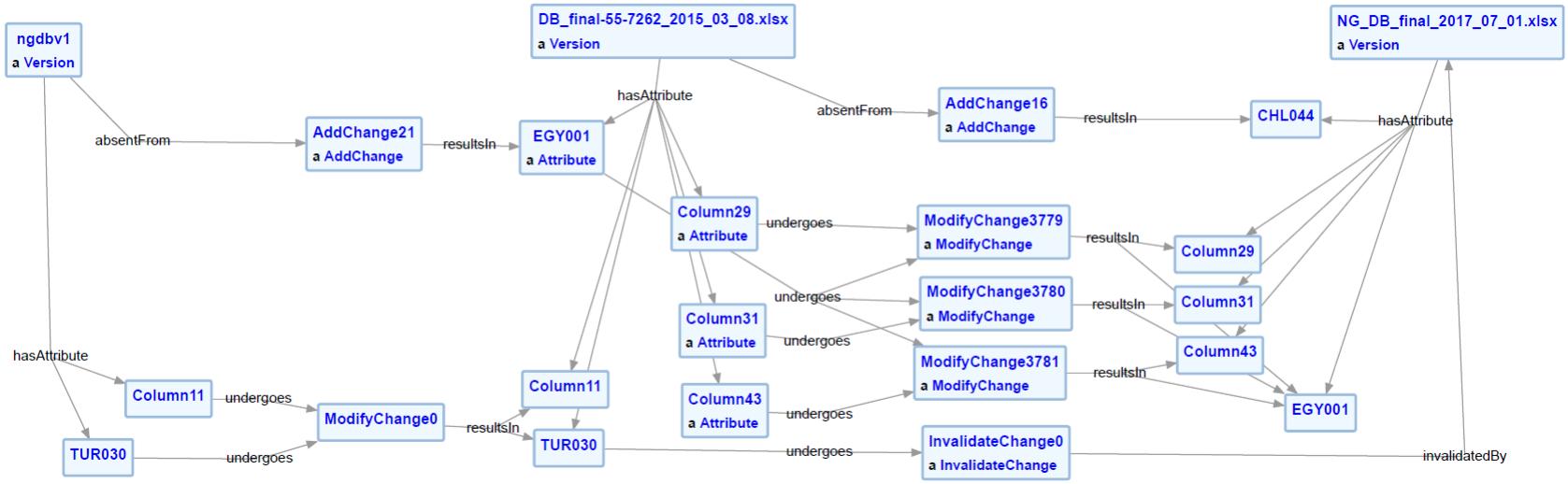


Figure 4.5: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications after the publication of the third version.

Section 4.3.4 to chain together versioning graphs. A change log was generated for each pair of consecutive versions in GCMD Keywords 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 4.6. The query used to extract the counts is found in Listing 4.4. 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.

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

```
8 } GROUP BY ?p
```

Listing 4.4: This query compiles the counts for each subclass of Change in a GCMD versioning graph

Changing the mapping method to account for the new namespace provides 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 4.7. 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. The **additions** in "8.4 to 8.4.1" in Figure 4.7 numbers almost a hundred, providing evidence that the trend of decreasing order of magnitudes may now continue as the

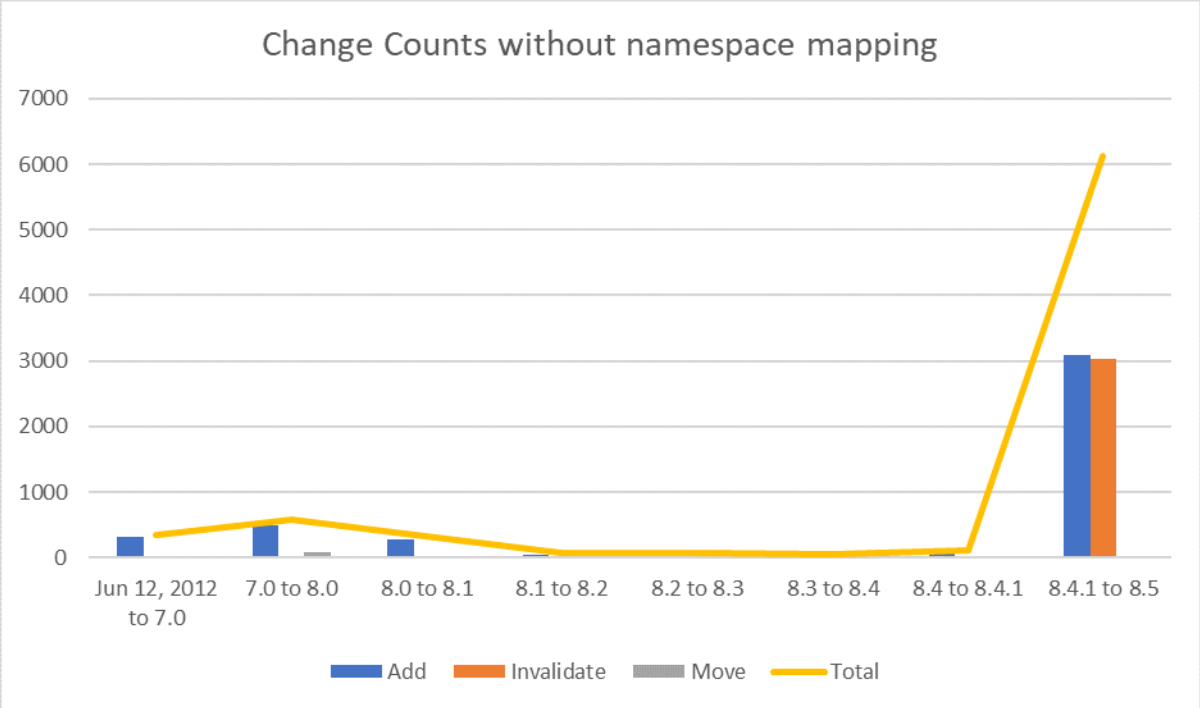


Figure 4.6: Add, Invalidate, and Modify counts in Version 8.5. The counts show change magnitudes and indicate that major and minor changes differ by orders of magnitude.

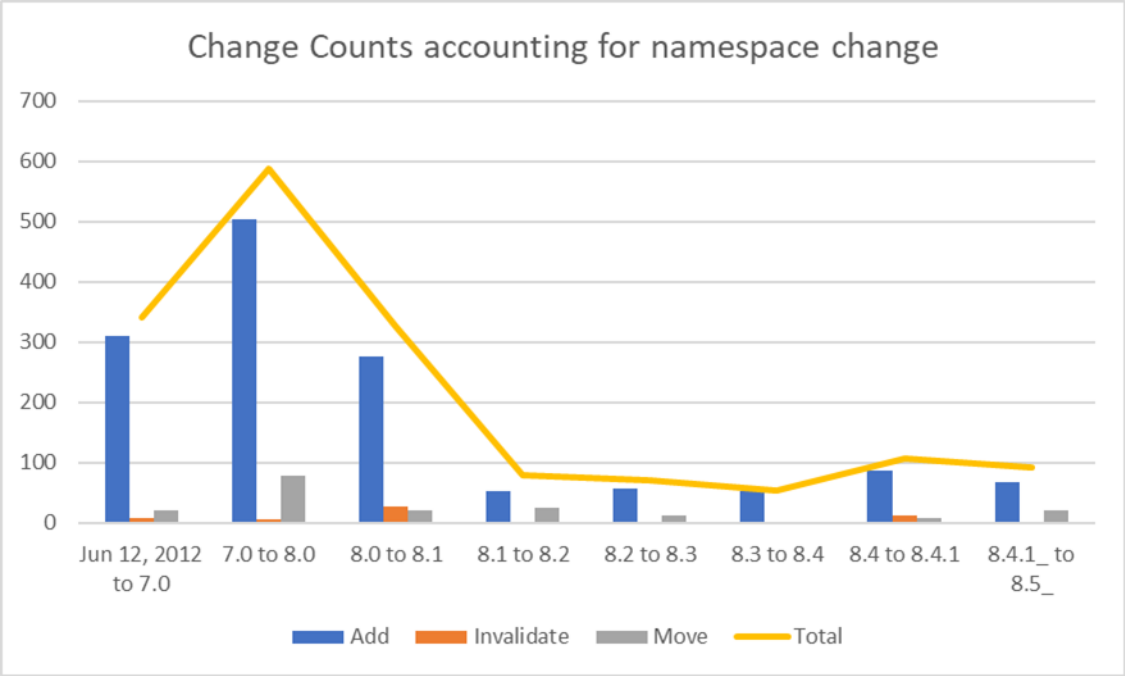


Figure 4.7: Add, Invalidate, and Modify counts ignoring the namespace changes in Version 8.5. The counts show change magnitudes appropriate for the identifier.

granularity of the version identifier increases.

4.5 MBVL

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 is unavailable, 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

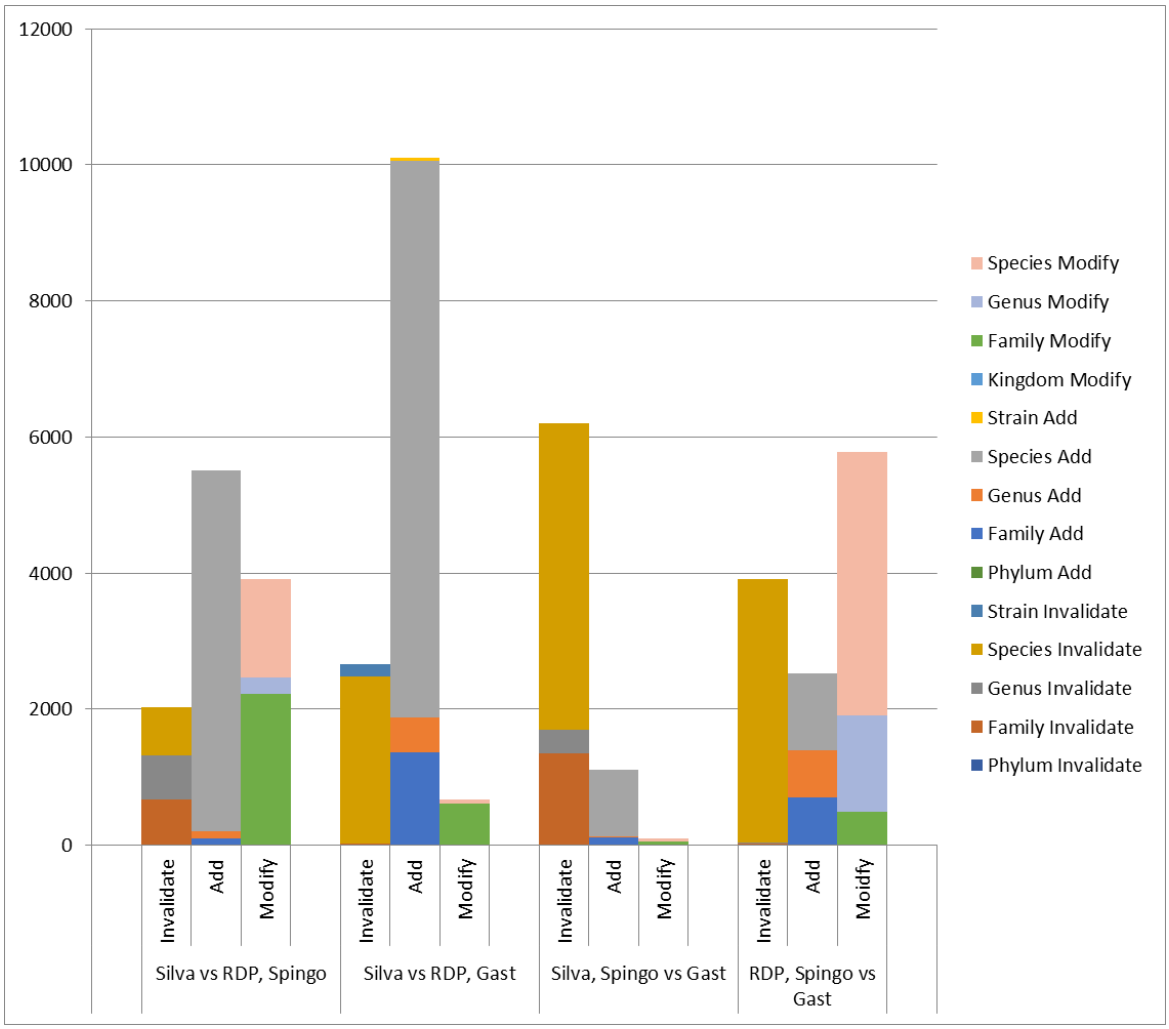


Figure 4.8: Compiled counts of adds, invalidates, and modifies grouped by taxonomic rank across algorithm and taxonomy combinations.

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 4.8 shows the changes detected when varying either the taxonomy or the classification algorithm. No comparison was conducted with different taxonomy and classifier since that would introduce too many sources of variability to differing results in a classification. 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 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. These groups organizes the data into a form to publish into change logs which are much less restricted when encoded using JSON-LD rather than RDFa. The resulting logs end up very large and sometimes do not load in a browser. The versioning graphs for GCMD Keywords seem to indicate that there may be a relationship between change counts and version identifiers. Finally, the MBVL data set demonstrates a case where versioning graphs can be used to compare the performance of different taxonomy/algorithm pairings.

CHAPTER 5
ANALYSIS

5.1 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.

5.2 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.

5.3 Implementation

5.3.1 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.

5.3.2 Change Log Analysis

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 4.2 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.

5.3.3 Version Graph

In Chapter 3, there is only one **attribute** on each side of the interaction. Figure 4.3.3, 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 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 4.3.4, 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**.

5.4 Version Identification

The versioning process discovered a discrepancy in the identifier assignment in the GCMD Keywords taxonomy. The original analysis was intended to determine if dot-decimal identifiers could be predicted using the change counts of the versioning

graph. Version 8.5, however, was named with respect to perceived taxonomy changes and did not consider underlying linked data practice revisions. The disconnect brings into question the accuracy of all prior names and any relationships observed between identifier and change counts. Non-matching identifiers would explain how 8.4.1 had more additions than any previous minor change but obtains a third bracket identifier. After accounting for namespace differences in version 8.5, the change counts is in the tens, resembling tallies of other versions in the same identifier bracket. Version name assignment based on producer perception and not on more concrete measures is concerning. An incomplete understanding in the amount of change between two versions can lead to flawed expectations during version migration.

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 4.7, 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.

5.5 MBVL Analysis

In Chapter 4, 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 4.8, 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 "unclutured bacterium". This analysis presents evidence that using the RDP taxonomy with the SPINGO algorithm will produce the most accurate classification results.

5.6 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 6
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.

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.

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.

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. Ap-

plying 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.

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

[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. 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.

[9] 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.

[10] 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.

[11] 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

[12] 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

[13] 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

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

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

[16] 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

[17] 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

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

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

[20] 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

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

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

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

[24] 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

[25] 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

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

[27] 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

[28] 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.

[29] 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

[30] 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.

[31] 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

[32] 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

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

[34] 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

[35] 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

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

[37] 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

[38] 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/

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

[40] 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

[41] 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

[42] 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

[43] 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.

[44] 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.

[45] 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.

[46] Y. Gil and S. Miles, "Prov model primer," April 2013, accessed: December 17, 2016. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/

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

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

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

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

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

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

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

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

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

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

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

[58] 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

[59] 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

[60] 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

[61] 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

[62] 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

[63] (2012, Jun.) Dcmi metadata terms. DCMI Usage Board. Accessed: February 8, 2017. [Online]. Available: http://dublincore.org/documents/2012/06/14/dcmi-terms/

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

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

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

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

[68] 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/

[69] 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

[70] 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.

[71] 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.

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

[73] 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

[74] "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

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

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

[77] 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

[78] 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/

[79] 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

[80] 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

[81] 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

[82] 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.

[83] 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.

[84] 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.

[85] 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

[86] 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

[87] 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

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

[89] 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

[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] 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

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

[94] A. Miles and S. Bechhofer. (2009, Aug.) Skos simple knowledge organization system reference. W3C. Accessed: January 19, 2017. [Online]. Available: http://www.w3.org/TR/2009/REC-skos-reference-20090818/

[95] 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

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