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

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 [1]. They provide necessary documentation which informs the transition to new methods and procedures [2]. They provide accountability for the value of a project's dataset when considering an agency's continued funding [3]. As Barkstrom writes in 2003,

> 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.

[4].

1.1 Defining Versions and Versioning

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," [4]. This definition's first implication is that groupings indicates an expectation that versioning requires multiple objects. With only one object, all metadata would be homogeneous and groupings would become unnecessary. Whether in the same or different groups, the relationship between entries informs the grouping necessary to organize the data collection. Figure 1.1 visually displays this concept with each file having a clearly defined grouping at

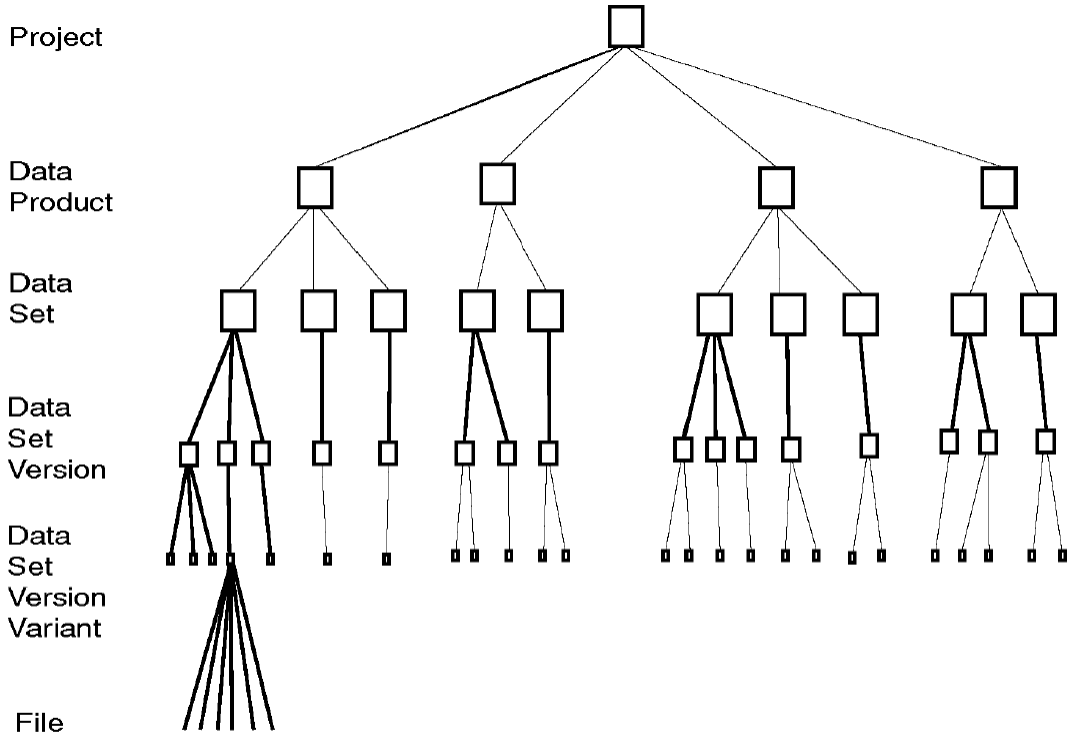


Fig. 2. Hierarchical groupings of files from data products to individual files

Figure 1.1: Visual representation of grouping hierarchy. From [4]

each hierarchical level. Another implication is that version changes can be determined by studying the difference in provenance between two objects. Changes in algorithms and coefficients may induce enough volatility into the data that a new grouping becomes necessary. However, this relies heavily on knowing which provenance changes correspond with resulting volatility. A new version of a data set can result from modifications to data downstream propagated through a workflow without significant changes to the immediate algorithm or coefficients. In addition, a deeper investigation would become necessary to quantify the amount of volatility introduced by these changes, and how much is necessary to induce a change at each level of the hierarchy. Therefore, a version definition cannot rely on provenance alone.

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

meaningful snapshot of a design object," [5]. Unfortunately, he does not further clarify what he means by semantically meaningful. It is, however, important to note that these snapshots are not of any design object, but the same one. A similar context connects each snapshot, meaning that they are versions as a result of sharing the same subject and setting. More specifically, these designs perform the same function in their application.

Combining these conclusions together, a clearer image emerges. A version is, at its core, a work, but it is not a singular work. Versions are members of a group which have common, but varying provenance. However, these members must also perform the same function as others in their group. This creates specific requirements as to when two objects can be considered versions of each other.

Notice that neither definition includes how to generate versions. The derivation, PROV Ontology's analog for a version, 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 because an activity generated it. However, looking more closely at the forces acting upon the version, they generate an object which becomes a derivation of another as a result of sharing the properties previously mentioned. The version identifier becomes applied using prior knowledge, not as a result of generation activities. That is, an activity determines whether an object is a version of another based on the state of that object. Therefore, versioning is the activity of identifying two objects connected by provenance and exposing the changes relating them, not of generating versions.

1.2 Identifiers

Data managers often impose a sequential ordering in conjunction with logical groupings to form an object's historical lineage [7]. As a result, they often employ version identifiers in the dot-decimal style where a decimal point connects together a series of numbers [8]. Whenever, a new version is made, it receives an identifier with one of the numbers incremented as seen in Figure 1.2. Changes to the left-most number often signify a more important change. Many software applications consider

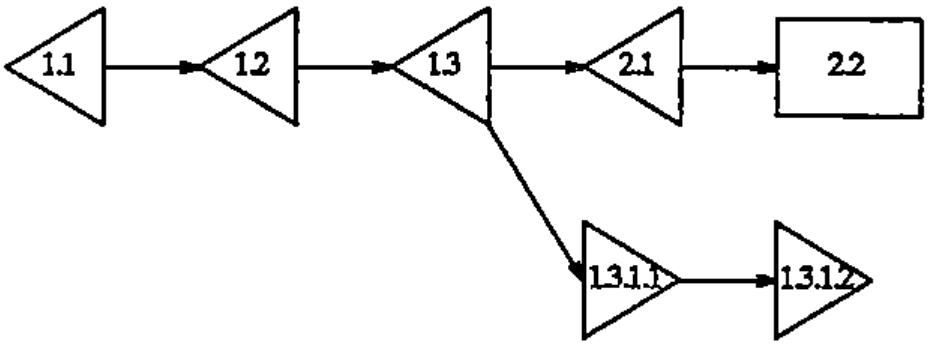


Figure 1.2: Commit history of an object in RCS with changes in the main line stored as back deltas and side branches stored as forward deltas. Figure 5 in [13]

the 3-number Major.minor.revision format to be a best practice 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 [9]. Most importantly, it groups together changes from the lower spectrum of minor or major change with those in the upper, more impactful, changes. It is, therefore, difficult to obtain a clear characterization of a version change without a longer series of numbers. This method of identification also has issues since small changes in the current data set may have larger implications for data farther down the workflow, which is not communicated by the identifier. In addition, version numbers capture the overall change of a data set, but users may not interact with collections that way. Data on a hurricane sometimes occupy only a small portion of a data set, and the files may not even be adjacent in the collection [10]. 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 [11]. Fischer, et al., demonstrate the importance of having a regular standardized version identification system, as it provides a mechanism to track errors being addressed [12]. The ability to link bug reports and versions thus gives producers and customers metrics to understand how developers fix errors and how long it will take. A common method used to address the distinction between versions is a human-readable change log, further discussed in Section 1.3.

In many ways, versioned objects resemble multi-edition books or documents. Digital librarians have faced many challenges when searching for an 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 [14]. Locators required a system to manage changes of old identifiers to new locations when people attempted to utilize references from print. This 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 [15]. The PURL used a centralized system that would translate dead links and redirect to a document's latest location. However, 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 1.3, DOIs represent the most suitable identifier used for citation in scholarly literature [15]. 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. However, the data set production and distribution cycle moves more quickly and reacts more sensitively to small content changes, including when data collection continues on after initial publication. This behavior becomes entirely too slow as data providers begin allowing users to dynamically generate data products from existing data according to their needs [16]. Some agencies have begun assigning versioned DOIs, but this has not become common practice. Other groups do not assign a new DOI, but reference to the latest release of the document or object [17]. As data sets continue to grow in size, it may become impractical to look towards data as the driving source for identifiers, as Proell and Rauber suggest that database queries may provide a more scalable means of citing datasets [18].

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 1.3: Table of predominant identifiers used in science. From Duerr, et al. [15]

The discourse on DOIs highlights the importance of understanding the limitations of particular identifier schemes. With respect to Figure 1.3, 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" [15]. 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. Further discussion on versioning models will be found in Section 1.4. This proves increasingly important as the ability to propagate relevant data change across autonomous systems then assures valid quality in interactions between domains [19].

Using identifiers to convey extended versioning information becomes more difficult with the adoption of distributed version managers like GIT [20]. 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

Figure 5.3: Benevolent dictator workflow

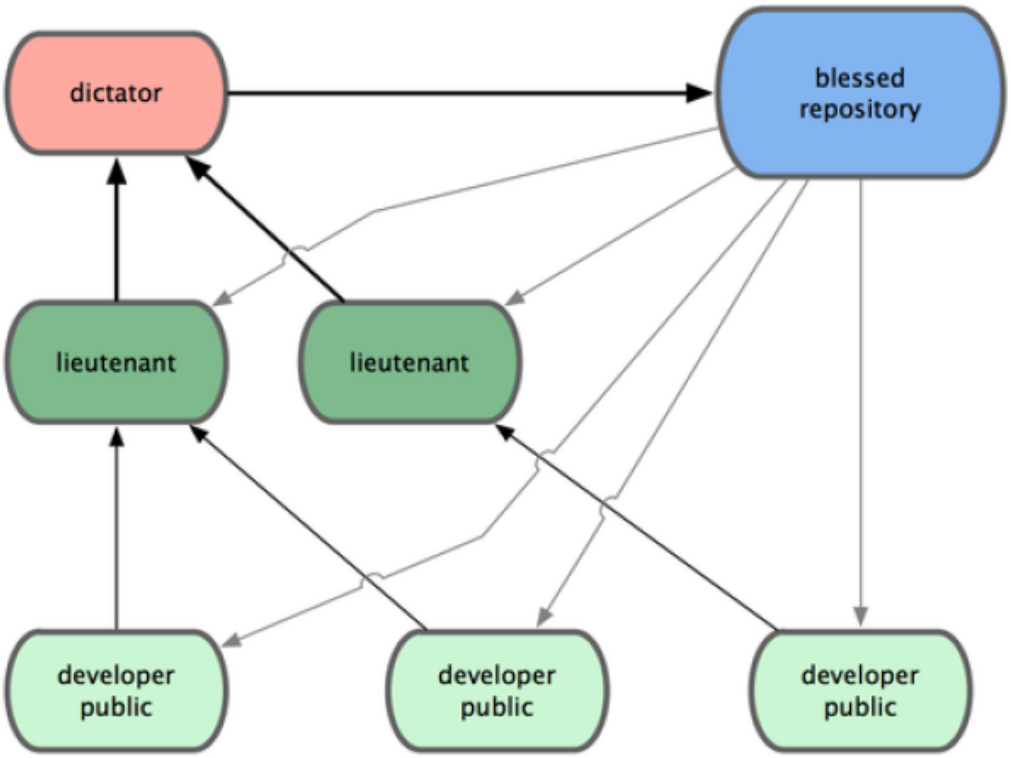


Figure 1.4: A distributed workflow to control for volatile versioning behavior. From [20].

follow a linear identifier scheme, this 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 1.4, 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.

1.3 Change Logs

Change logs, artifacts resulting from the versioning process, play a major role filling in gaps between versions. They document changes and explain, in human language, the motivations behind these modifications [21]. 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. They allow developers to link bugs and errors with their corrections in new versions of the code [22]. This gives insight into motivations behind particular design decisions. It also provides 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 [23]. This becomes particularly significant as data freshness plays a growing role in successful system function [24]. However, 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 [25]. 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

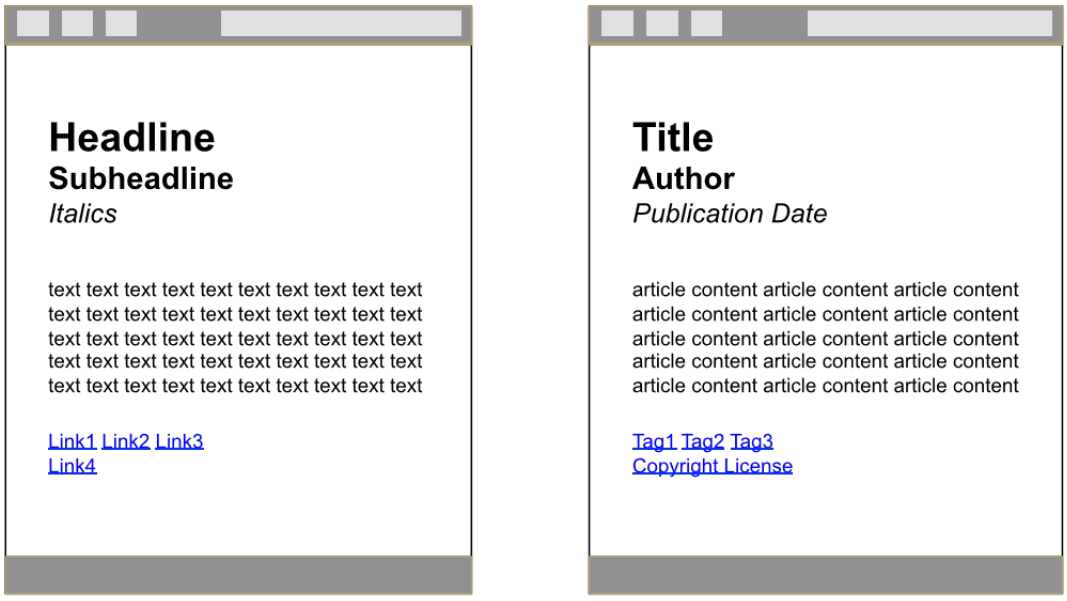


Figure 1.5: Illustration of the difference in what autonomous systems see when crawling a web page and what humans see when reading the same material. Figure 1 from [27]

any of the content within logs more than understanding they contain text. The transition between different versions of large datasets is then left largely up to the human user's ability to understand and process the modifications mentioned within the change log.

1.3.1 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. [26]. Linked data technologies utilize web ontologies to standardize terms and semantics across domain applications. Automated agents can reason over a graph formed by linking these terms. Encoding HTML documents with linked data better unlocks the content stored within. Figure 1.5 illustrates the semantic difference between what web crawlers and what

humans see when they consume web pages. People intuitively understand that certain strings represent meaningful information based on location and style. RDFa seeks to encode that understanding natively for effective machine consumption. Extending this approach into publishing change logs, will allow linked data to capture the metaphorical meat of change content.

The implementation requires changing publishing practices from plain-text documents to something structured-data compatible such as HTML. The change also has the added benefit of making the logs available on-line, and thus, more openly accessible to data users through the utilization of web based search engines. Large companies such as Google have already begun equipping their web crawlers to consume structured data such as RDFa from web pages. RDFa has already had significant success in adoption across a variety of web publication platforms and eases the search for their content [28]. However, the design of RDFa focuses on describing the web page's content through markup [27]. The underlying or resulting versioning data model may not conform with the format of content presented in the change log. This 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 [29]. Because it stores data separate from visible content, JSON-LD does not need to adhere with the constraints of visible content. However, every linked data triple must be explicitly defined, meaning that resulting documents may likely be much larger than their RDFa counterparts.

1.4 Modeling

Initial research in version modeling occurred with databases, studying ways to track and migrate schema changes. Capturing periodic snapshots or copies becomes

unfeasible with increasingly large centralized database systems. This gave rise to the first temporal databases where schemas included time and dated transactions modifying the schema [30]. This provides the database a method to transactionally rollback the environment to recreate a search. As databases grew more complex, the intricate relationships between objects made rollbacks more difficult. This results from the need to manage the time instances of realization, storage, and validity. The datum becomes realized at collection, then stored upon entry into the database, and finally valid until the present or new data replaces it. Klahold, et al., introduced using abstract versioning environments to separate the temporal features and organize them into related groupings [31]. As publications more often include citations to data, researchers adjust to enormous modern databases with new methods changing queries rather than transforming schemas [32] [33]. This method, however, relies on an abstract model capturing data changes to inform query modifications.

Data models allow the capture of complicated relationships between data objects within a system without needing to physically look over sizable datasets. The Proof Markup Language, one of the first semantic models to capture provenance information, expressed these relationships using inference reasoning through traceable graphs [34]. Many data models include versioning concepts in their specifications, but they are often provenance models which describe the sequence of events that lead to the construction of an object [35]. However, as discussed previously, versioning seeks to expose relationships between objects which may not describe how to construct either one. The Health Care and Life Sciences (HCLS) Interest Group recently released a model which may provide a solution when used in conjunction with other identifiers [36]. Their model, shown in Figure 1.6, separates the concept of a dataset into three groupings. The highest level summarizes the data as an abstract work, perhaps better described as a topic or title. This data topic can have multiple versions as it changes over time. The version can then be instantiated into various distributions with different formats. Compare this to the diagram in Figure 1.1 which features more tiers. In this case, the basis for diverging groups is changes in identifier and physical formating of that version. This model also demonstrates how it can better communicate scientific similarity than an identifier. The model

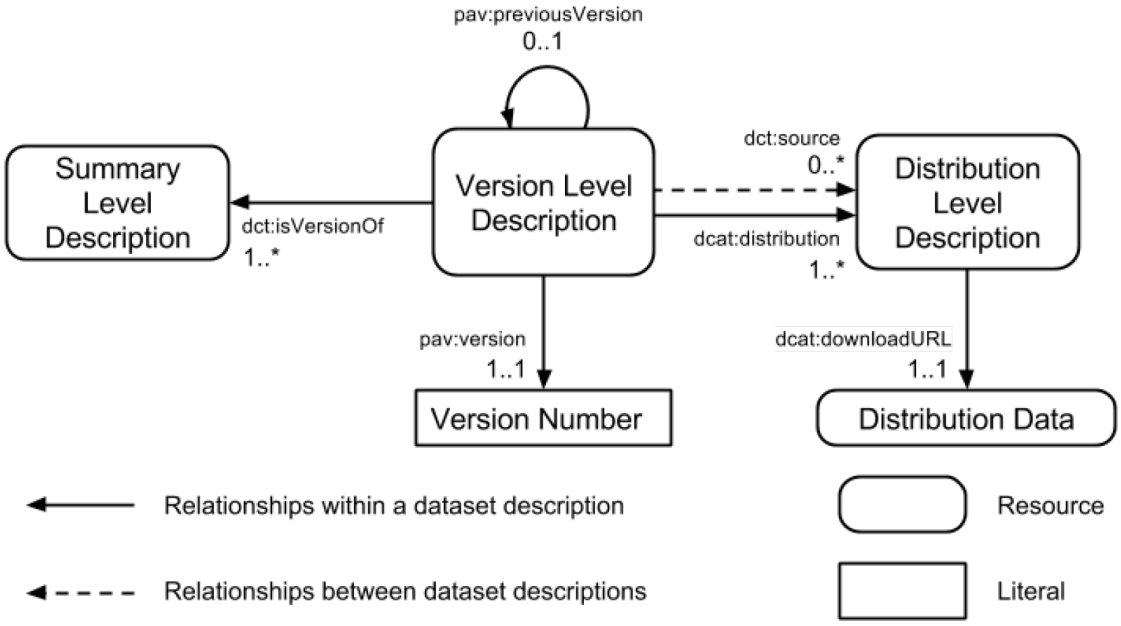


Figure 1.6: Data model from the Health Care and Life Sciences Interest Group separating data into three levels: works, versions, and instances. From Dummontier, et al. [36]

logically links together, under a single version, representations of that data adjusting for physical perturbations.

The HCLS data model captures version information using the Provenance, Authoring, and Versioning (PAV) ontology [37]. As explained in their document, PAV addresses versioning as a result of disagreements with other ontologies' definitions such as PROV. They enumerate specific situations relating to health care data requiring broader conceptual coverage. As a result, their model provides basic properties to identify a previous version and its corresponding identifier. They do not provide methods to identify motivations or discuss the role attribute differences effect changing versions. The existence and widespread adoption of change logs demonstrates the importance of this information in understanding modifications between versions. However, PAV does adopt a state-based perspective regarding versioning since it only seeks to identify a previous version without attributing a responsible activity.

The PROV data model is a World Wide Web Consortium (W3C) recommendation which defines linked data properties to capture data provenance [38]. PROV

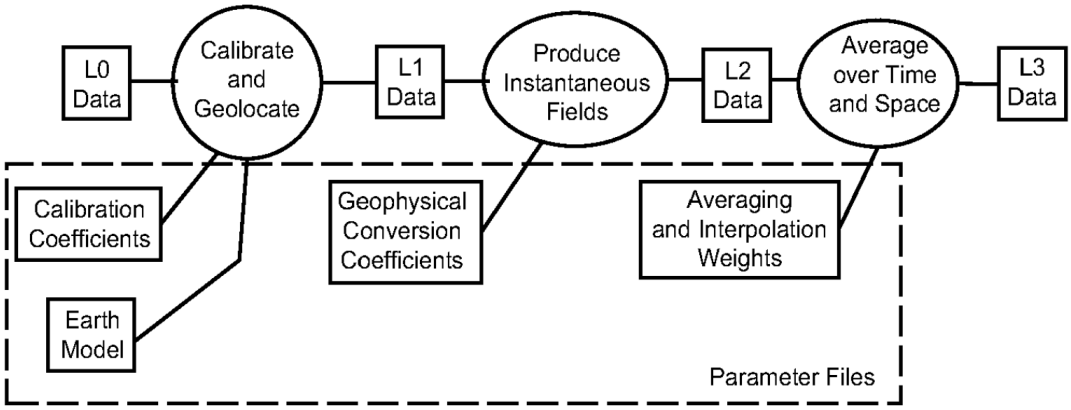


Figure 1.7: NASA organizes its data into three levels depending on the amount of aggregation and the distance the data is removed from the original sensor measurements. Figure 1 from [4]

has played a significant contribution in maintaining the quality and reproducibility of datasets and reporting in the National Climate Assessment (NCA) [39]. This implication signifies that there is an increased likelihood of adoption through other scientific fields as a result of this reporting. The Global Change Information System, which houses the data used to generate the NCA, uses PROV to meticulously track the generation of its artifacts and results as they are used in the report [40]. This means that not only does the data have a traceable lineage to verify quality, but the content of documents can have the same verifiability [41]. While a subtree of terms deal with versions, the ontology takes an activity based view of data. This makes sense since it seeks to explain what activities led to the creation of an object. This focus on activities results in a very shallow definition of methods to capture entity-to-entity relationships within the model.

In his discourse on version management, Barkstrom introduces NASA's workflow model as seen in Figure 1.7. It describes the formal stages of processing to turn a raw remote sensing signal from satellite instruments into global aggregate summaries [4]. Understanding this model reveals that changes to either the algorithms or parameter files will force a change in the resulting data. This explains the new result's provenance, but it would not capture the differences between these objects. The key to understanding version modeling lies in recognizing provenance and versioning are intimately entwined, but seek to explain different properties about data

objects.

Klein and Fensel determined versioning systems needed to capture compatibility as a major characteristic while performing a study of different ontology versioning methods [42]. There are two types of compatibility, 'forwards', which explains how to modify old data to work with new content, and 'backwards', which allows new data to interact with old data. Knowing one type of compatibility does not always entail the means of deriving the other. Having full forwards and backwards compatibility allows interoperability between a variety of applications which may use different versions of the same ontology. However, it often depends on the application as to whether full compatibility is necessary. Once concern which is very rarely discussed is the compatibility of files and objects which do not change at all. Take, for example, some biological ontologies, which must update terms and definitions to better capture the field [43]. They mention very little on how to migrate terms which do not change, but these can contribute to a majority of copied content and pose significant scaling issues.

1.5 Data Versioning Operations

Versioning systems come in a variety of different forms as a result of the assorted applications they must manage. Experimental biology data require tracking as they pass through a scientific workflow [5]. Libraries curate multiple editions of the same work, sometimes with significant revisions [2]. What such activities demonstrate is the wide range of settings and expectations under which versioning systems operate. While this work cannot hope to explore all these applications, every system studied shares three common functions: addition, deletion, and modification. These three operations plausibly form the core set of relationships when versioning.

Literature surveys often expect versioning systems to interact with data uniformly because they are asked to perform the same functions [5]. However, different data sets may utilize each of the three core operations at different rates [44]. This helps 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 dataset, but most major provenance ontologies do not heavily feature these relationships. Versioning information, of course, does not give insight into the origins of an object, and therefore, would not be expected to play a major role in provenance documentation.

While data addition and modification remain fairly uncontroversial, there is a mild division between practical and theoretical approaches to data deletion [45]. 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. The software versioning manager GIT uses a method of compressing older data to conserve space without deleting the data [46]. 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 [3]. 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 [13]. This mostly provides an organizational role in development by allowing developers to experiment without contaminating a stable software release. Figure 1.8 models this, showing versions C3 and C5 in branch iss53 before being merged back into the production line as C6. It 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. It does not introduce any new relationships but allows the tool to operate more smoothly. Clotho, a versioning application managing versions at the block level, coordinates constrained spaces using intermittent

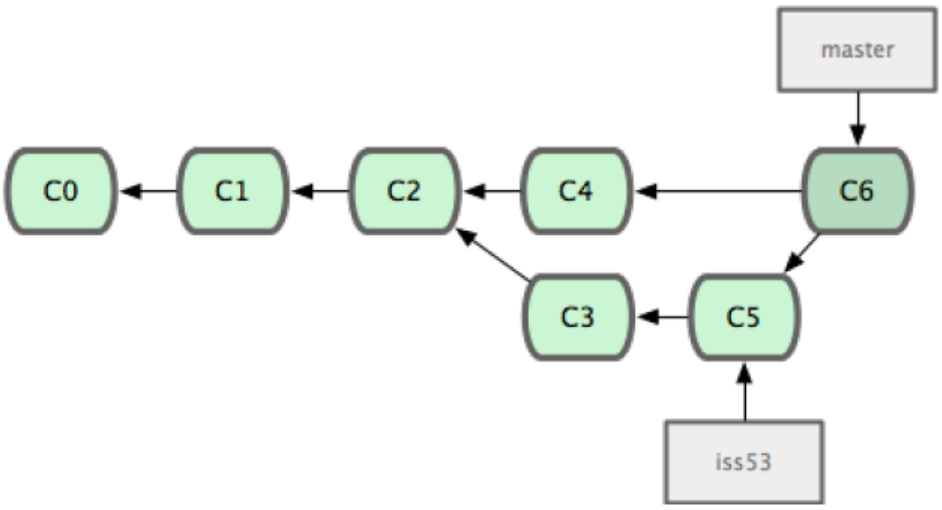


Figure 1.8: Example of a commit history with branching stored in GIT. Figure 3.17 from [46]

compact and un-compact methods when retrieving or storing old objects [45]. Likewise, many version control tools include functions to display the versioning tree, but this is also an ease-of-use function [9].

1.5.1 Types of Change

Barkstrom uses the ability to scientifically distinguish between two data sets as a criteria for major divisions among groupings [4]. At lower levels, he notes that science teams can no longer discern scientific differences between data sets. Instead, they observe that 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. These terms were chosen carefully as they reflect the three value dot-decimal identifier system. However, 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, and quantitative analysis on versioning operations will provide quantifiable evidence in characterizing the impact of a change.

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. This conflict in view illustrates the data consumer-producer dynamic. In general, data producers control the versioning methods, but data consumers determine the classification of a data change. Producers tend to use versioning systems to ensure data quality of service through audits and recovery tools [3]. 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. This document's approach studies add, invalidate, and modify counts to quantify the impact of changes and how they relate to the producer's observations. As previously mentioned, different systems utilize the operations at varying rates so absolute cutoffs will be unlikely in comparison to relative results.

1.6 Thesis 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. A model is developed to provide a foundation in regularizing the versioning process and communicating that in a machine-computable manner using linked data. For this purpose, we compare two methods in micro-data with JSON-LD expected to provide better performance. These relations cannot be communicated by current provenance models' semantics. Furthermore, the model provides a quantitative basis for determining meaningful version identifiers following a numerical system which

has previously only been performed using qualitative methods. Through sub-classing concepts within this versioning model, versions can provide detailed comparisons to support experimental designs and decisions.

CHAPTER 2
PREVIOUS WORK

The data versioning landscape produces a variety of different approaches and standards towards change capture. However, massive centralized data stores have become more prevalent as data distribution methods advance [47]. Collection into larger unified repositories will likely require a multi-tiered approach to synchronize the varied practices [48]. Baker notes that differences depend on the sociotechnical distance of a repository from the data's origin [48]. Local stores closer to the collection site better understand data capture conditions, but must also adapt to changing environments. This work provides a basis for understanding the formal underlying properties which will allow consistent versioning practices. Science agencies and organizations are only beginning to formally codify and standardize methods to capture and publish lineage information [49]. This seems to be a recurring cycle of attention with rapidly developing technologies such as the grid or parallel computing [50]. The CERN grid for the Compact Muon Solenoid experiment carefully developed necessary processes which allow references by multiple users to the same file without copying that file across the grid [51].

2.1 Provenance

A number of linked data models include versioning concepts such as the Open Provenance Model (OPM) [52]. 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 [53]. 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 [54]. The property WasDerivedFrom constitutes a core concept in the model and marks the reliance

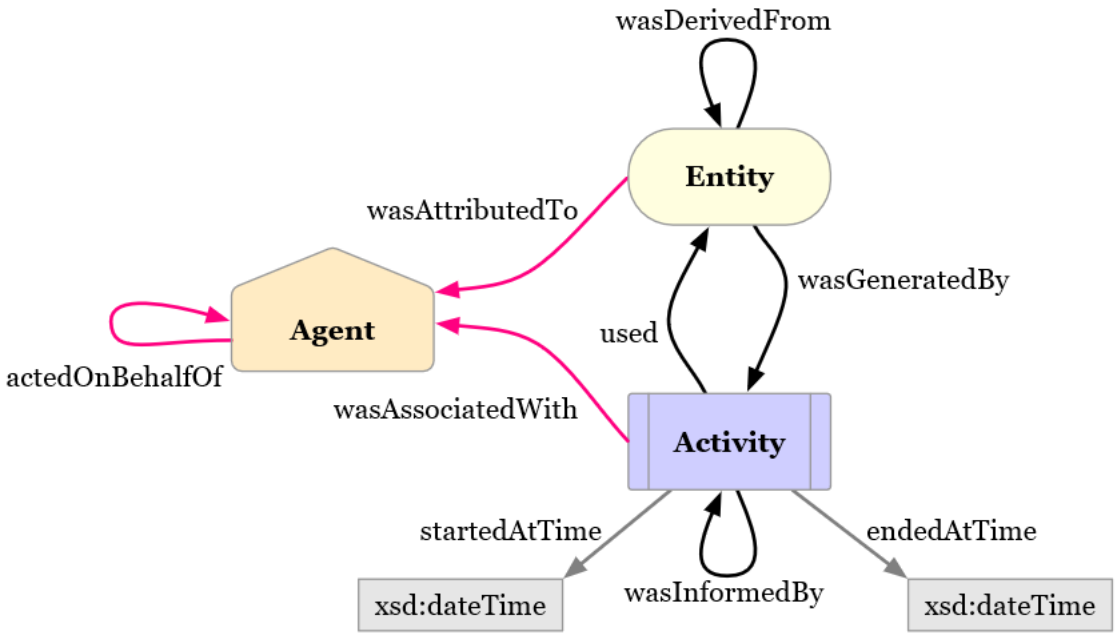


Figure 2.1: Diagram of the PROV Ontology. Figure 1 from [6]

of one object's existence on another object. For a large part, this encompasses the engagement which provenance models view versions, without further need to explore the derivation's content.

PROV, a World Wide Web Consortium (W3C) Recommendation, delineates a method to express data provenance in a more compact form as seen in Figure 2.1 [55] [56]. The recommendation uses a conceptual model relating activities, agents, and entities to describe data production lineage [57] [58] [59]. 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. The conceptual model's expression occurs through the PROV Ontology (PROV-O), which can be conveyed through various resource description languages [60] [61]. The ontology is further formalized into a functional notation for easier human consumption [62] [63]. 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 [64] [65]. Komadu, a framework developed to alleviate workflow integration, improves upon its predecessor, Karma, by no longer utilizing global context identifiers that were not necessarily shared throughout the workflow. [66].

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 to the version addition operation, must result from a *prov:Activity*. However, activities play a much less active role in versioning since object comparisons instead expose changes. The property creates a relationship between entities and activities, but such a connection may imply that perturbations in the activity resulted in changing the version. Changes could also result from modifications in the input data, leading to an entirely new generating activity rather than a modified one. *Prov:Invalidation* likewise makes a similar connection between activities and entities. This means that PROV-O does not have the direct means to communicate the addition and invalidation relationships which exist in our versioning context. Since we previously establish a state-based view between versions, a more contextually appropriate property should connect two objects together. Continuing, *prov:Derivation* does relate 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]. In the Marine Biodiversity Virtual Laboratory (MBVL) dataset's case described in Section 3.4, none of these three assertions hold true. The process simultaneously considers all four versions so one is not transformed into another as would a sequential set of versions. Additionally, since we do not know which version is the best, we cannot consider any 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.

The Provenance, Authorship, and Versioning (PAV) Ontology is, "a lightweight vocabulary, for capturing "just enough descriptions essential for web resources representing digitized knowledge" [37]. 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 [37]. 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" [67]. 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.

The Schema.org *schema:UpdateAction* largely reflect the relationships adopted by this work, and are defined as "the act of managing by changing/editing the state of the object" [68]. The remaining subclasses include the *schema:AddAction*, *schema:DeleteAction*, and *schema:UpdateAction*. The function of these terms is to provide a means to supply searchable web pages with standardized micro-data. As a result, they orient their properties towards definitions and characterization but do not provide the right structure for a standardized change capture model.

2.2 Provenance Distance

With increasing complexity, data workflows have developed in such a way that even subtle changes have serious implications for other parts of the workflow [69]. This observation makes change impact difficult to measure, but one insight begins with provenance's role in workflows. Provenance can give great insight into a data object's future performance such as the ability to predict disk usage based on the lineage of a data object [35]. Efforts have also been made to summarize provenance representations to improve consumption [70]. Changes to the process creating an object signals the development of a new version. Therefore, studying the magnitude of this deviation should give some idea into the resulting object's impact. This idea, 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 2.2. It 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

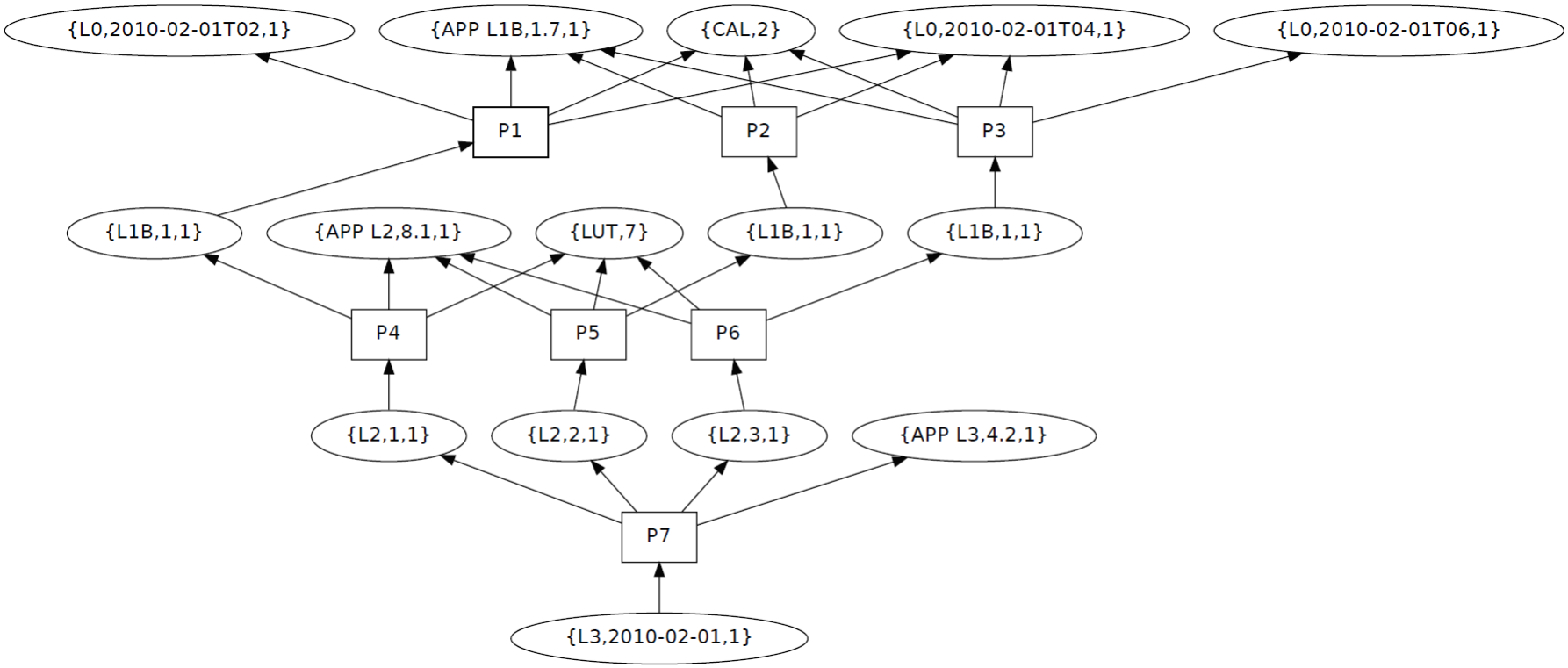


Figure 2.2: Provenance graph of a Level 3 data product, showing the inter-relations between different data products in generating the final product. Figure 2 from [69]

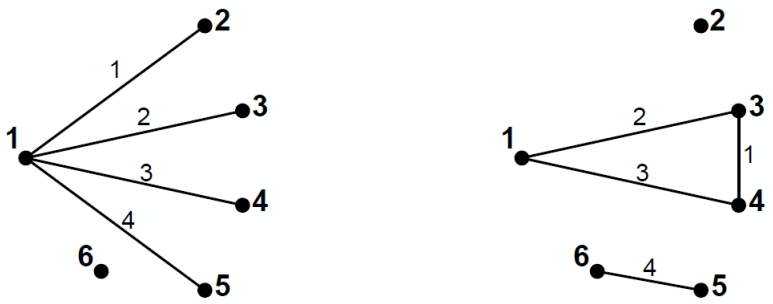


Figure 2.3: The labeled graph on the left transforms into the right graph under two edge edits. Figure 2 from [73]

reduces to similarity measurement. When calculating this measure, algorithms determine how far two graphs are from being isomorphic [71]. Node labeling simplifies this process by providing nodes which must match together, and greatly reduces the complexity from computing generalized graphs. Graph Edit Distance, counting the edits necessary to transform one graph into another, provides a quantitative measure to associate with this process [72]. Some variations count edge changes [73].

In Figure 2.3, 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. This kind of analysis resembles comparison measures employed in determining semantic similarity [74]. However, 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.

[69]. L-number is shorthand for the level system featured in Figure 1.7. 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 [75]. However, these procedures focus primarily on quick retrieval and efficient storage instead of capitalizing on the latent information accessed by reasoning across data set versions [76]. The distance measures previously mentioned rely solely on provenance graphs to compute results, but this is obviously insufficient. When considering the provenance of a data object, methods only consider the activities and entities that took an active role in the production of it. A new version of an object has a familial relationship with its previous versions, but in most cases, they do not take an active role in its generation. Without detailed change information, determining the difference between two data objects in a metric beyond broad strokes becomes difficult, if not impossible.

As per our definition of 'version', objects must have common provenance, and the more similar they are, the more meaningful the results from versioning methods. Provenance distance provides a means of determining how reliable versioning results are given a greater adoption of provenance graphs. Measuring a change's impact with accuracy comparable to a change log requires a more detailed understanding and description than provenance can provide [77]. 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.

2.3 Mapping

Data managers primarily use one of two methods to store data versions: snapshots and deltas. The snapshot method makes periodic copies of the data's state at a point in time. While storing and retrieving these snapshots can be very quick, they require significant amounts of space to maintain. The software manager GIT employs this method and Figure 2.4 demonstrates an example storage space for multiple versions [46]. The squares with dotted outlines indicate unmodified files, which the system stores as pointers instead of full objects. In addition, GIT compresses and separately stores very old versions which are unlikely to be accessed. This versioning style may not be ideal for larger or often modified data sets as the size

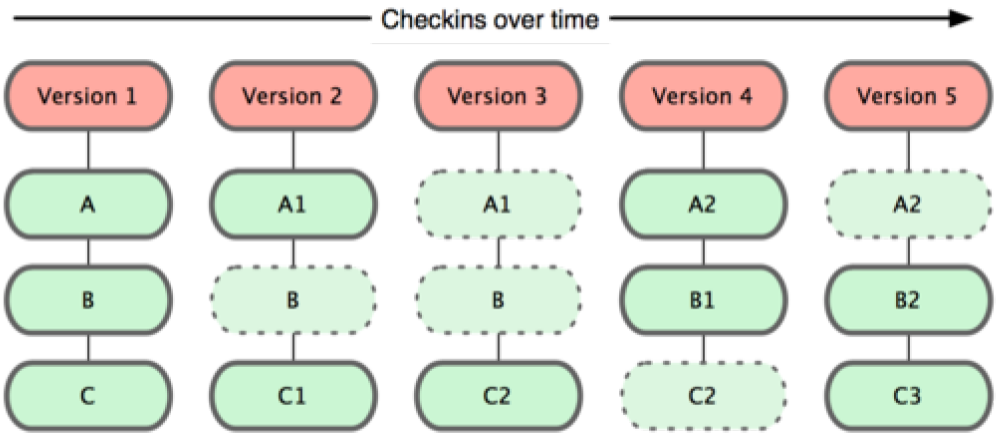


Figure 2.4: GIT stores changes in the repository as snapshots of individual files. Figure 1.5 from [46]

requirements will quickly grow unmanageable. However, for many library or catalog environments, they cannot predict the target volume a user desires and must prioritize availability [78] [10]. Some methods like the inverted file index have been developed to balance space and retrieval performance on web documents, especially since wikis and news feeds have grown in deployment [79]. Searches over these text media may require execution on older archived web pages.

The delta method entails calculating and storing only the differences between one version and the next. Back delta variations store a snapshot of the most recent version and compute deltas towards older releases. The forward delta variation stores the oldest data's snapshot and has deltas going forwards. This method uses the minimum amount of space but trades it in for computation time to recreate any given version. Particularly long running versioning systems occasionally save an intermittent snapshot to cut down on this processing time. The setup proves ideal for data sets which prioritize service to their most recent versions [8]. Because change documentation captures information between version objects, they most resemble differences calculated by the delta method.

Properly detecting changes in a system's files allows file managers to correctly group them into versions as seen in research conducted by the Atmospheric Radiation Measurements (ARM) group [80]. Difference or diff applications must first properly map data between objects and align them for comparison. Many text-

based data sets rely on well-established algorithms to perform this alignment [81] [82]. Sequential scientific data largely avoids this problem since developers already know the files or objects they replaced. However, users do not have this advantage and system managers are starting to recognize the difference in versioning usage patterns between users and producers [83]. Mayernik, et al., probably gives the best description saying, "Prospective records document a process that must be followed to generate a given class of products whereas retrospective records document a process that has already been executed" [49]. While producers take a retrospective approach to version usage, consumers of new versions must take a prospective view, adapting to new changes. This indicates that the orientation of versioning information reflects the imagined customer of that data.

CHAPTER 3
DATASETS UTILIZED

3.1 Noble Gas Dataset

The "Global Database on $^3He/^4He$ in on-shore free-circulated subsurface fluids" is a tumultuous database [84]. 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.

3.2 Copper Dataset

The Paragenetic Mode for Copper Minerals database became available through collaboration with the author's lab to create new methods of visualizing mineralogy relationships [85]. 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 4. The data from this work is also more processing friendly, making it agreeable to automatic change log generation.

Table 3.1: 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

3.3 GCMD Keywords

The Global Change Master Directory (GCMD) is a metadata repository used by NASA to store records of its available data sets [86]. They employ a set of keywords to make NASA Earth Science data sets searchable. These words tag and label datasets into strictly defined categories [87]. 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 [88]. 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 [89].

3.4 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 [90]. 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 3.2. However, this data is not available to the author, and only the list of species are known, forming the first data set in this

Table 3.2: 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

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. This produces four data sets, each using the same grouping identifiers and having the same size in each group. This means that the only difference between the data sets are the ranks assigned to each group.

CHAPTER 4
CONCEPTUAL MODEL

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

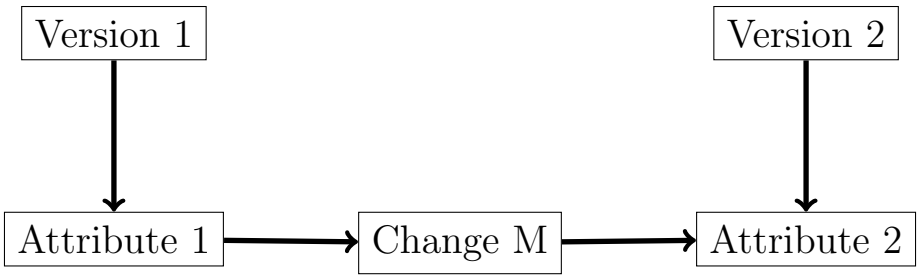


Figure 4.1: Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2

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

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

4.1 Modification

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

Notice that the model captures neither the change's magnitude, nor the values of the attributes involved in the change. These are not included because too much domain knowledge will be required to interpret the significance of the value. In addition, the model would essentially begin storing a copy of the dataset, leading to space and redundancy concerns. If an application needs to include this information, they can be added as a property of the attributes involved, but this is outside the scope of this thesis. Simply knowing that the attribute has changed provides valuable information to identify the relationship between the two attributes from a versioning standpoint. Sometimes it may be necessary to distinguish between modification changes. For this reason, the *vo:Change* concept in this relationship may be sub-classed to differentiate between different kinds of change that map one attribute to another. For example, in order to distinguish a modification in which the units of a measurement changes, a *UnitChange* concept, which sub-classes the *Modify* change concept, can be used to connect the attributes from two different version.

The *modification* relationship is simple enough to understand, and a couple of well developed alternatives exist. Schema.org supplies a subclass of the *schema:UpdateAction* labeled *schema:ReplaceAction*. This concept has two properties, *schema:replacee* and *schema:replacer* which indicates that a new object replaces an old one. This provides a structure very similar to the one found in Figure 4.1 with the only difference being the relationship between Attribute 1 and Change M reversed. While this difference is subtle, it demonstrates a disagreement in view as to how change works in a system. Schema.org's perspective looks at a single replacement action in isolation and models that quite well. However, this versioning model views change as a quantity that flows through attributes from one version to another. Unlike ReplaceAction, ModifyChange does not attribute changes to an actor, but rather it acts as a medium to convey change between objects.

PROV also has a property called *prov:isDerivedFrom* to convey, "a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity," [6]. Changing this property into a qualified property then results in a form very similar to the

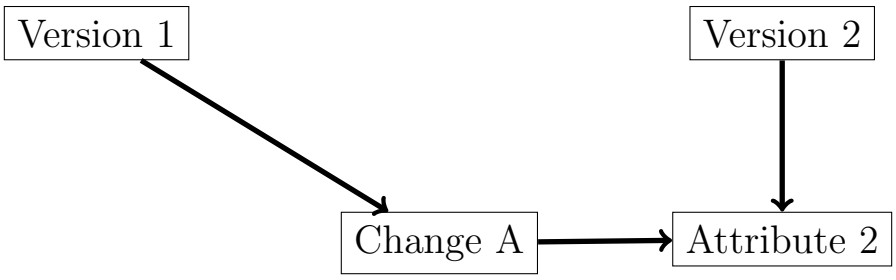


Figure 4.2: Model of the relationships between Versions 1 and 2 when adding an Attribute 2 to Version 2 as a result of Change A

construction in this model. It gains the advantage of turning the property into a concept that can be instanced and described easily using other ontologies. The primary discrepancy between this versioning ontology and PROV-O is that Version 2 no longer needs to depend on a previous entity. PROV assumes derivations to be part of a sequence, but situations exist, as seen in a later chapter, where research can create multiple simultaneous versions where an ordering does not matter.

4.2 Addition

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

In comparison, the *schema:AddAction* from Schema.org considers addition as an activity performed by an agent. This view of addition does not take into account the prior state of the collection being manipulated. As a result, the proposed model for *addition* should provide a better context for the relationships between

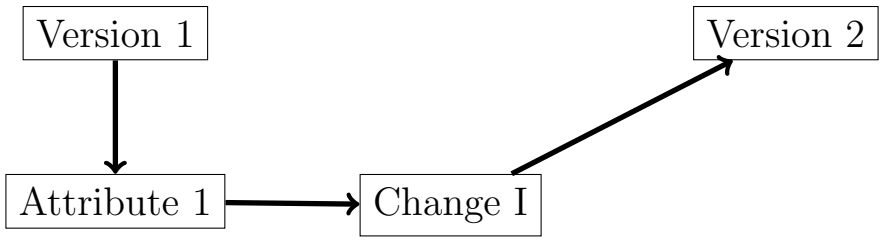


Figure 4.3: Model of the relationships between Versions 1 and 2 when invalidating Attribute 1 from Version 1 as a result of Change I

objects when an attribute is added. The corresponding concept in PROV-O is a *prov:Generation*. However, it relates together activities with entities, meaning that it could not relate together two versions or a version and an attribute. This property also does not make sense because, once again, no activity is producing the versioning relationship. Although, if we attribute new additions to the activity which generates Version 2, a relationship using *prov:Generation* can be formed. This formation, however, would also apply to other attributes added to Version 2, and convolute the role the activity took in an attributes generation. In the versioning model proposed, each added attribute connects to a particular change instance so that they can be individually described.

4.3 Invalidation

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

Related concepts include Schema.org's *schema:DeleteAction* and PROV's *prov:Invalidation*. PROV defines the property as the "start of the destruction, cessation, or expiry of an existing entity by an activity," but since nothing actively creates versioning relationships, this property is inappropriate for this application [6]. There is no activity to credit with removing the property in this comparison, and the absence results

from a state-based property. Schema.org defines the *schema:DeleteAction* as, "The act of editing a recipient by removing one of its objects," [91]. This definition assumes that an actor will perform an action upon an object or collection, removing one of its members, producing a new result. This follows the provenance mentality, but when versioning, the resulting object has already been created. Cases exist, as shown in Section 5.2, where an attribute is not included in a new version, thus indicating that it has been removed. However, the most important point of contention in this definition is the idea of removal and deletion. Although an object is no longer valid, it does not mean the data has been deleted. Invalid data often gets removed, but this is not always the case. As a result, naming the concept invalidation is more general and inclusive.

CHAPTER 5
VERSIONING TABULAR DATA

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

5.1 Validate Comparisons

The first step is to establish that the two objects being compared meet the requirements of being each other's versions. For the Copper dataset, the objects result from a compilation effort from the same sources of data. An entry recording the source material for each row in the table accompanies each reading. As a result, these entries were used to determine that they share similar provenance. They will also both be used as data inputs at the same step to generate data visualizations. The datasets become interchangeable within the context of executing the workflow. Determining whether the Noble Gas files are versions is a more challenging activity. From provenance graphs like the one constructed for the entry CAM001 in Figure 5.1, the entries share a common source document. Likewise, a significant portion of the other entries also share sources from the same body of work. The ones that do not are new or removed entries. As there was no personal involvement in the use of this data set, determining whether they can occupy the same workflow step requires actually looking into the files. Investigating column headers and the accompanying

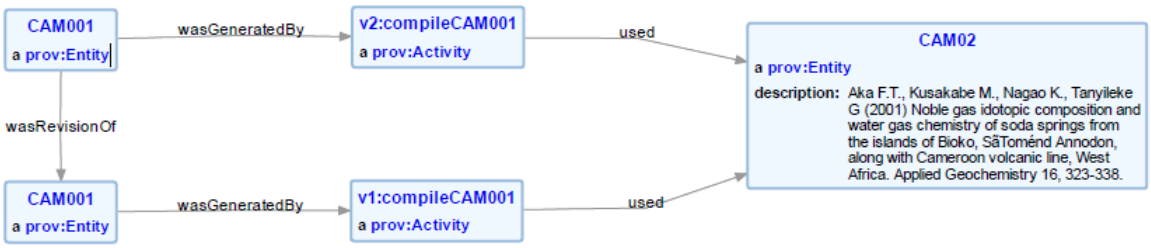


Figure 5.1: Provenance graph for the CAM001 entry of the Noble Gas Database. Other than the labels, the structure of each data object is very much the same.

documentation reveals that they report many of the same measurements. Some amount of leeway is given in this assessment as having every quantity match would mean that the two files aren't versions, but the same object. This process helps to determine the confidence that using the model to encode the discovered changes will result in a meaningful versioning graph.

5.2 Form a Mapping

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

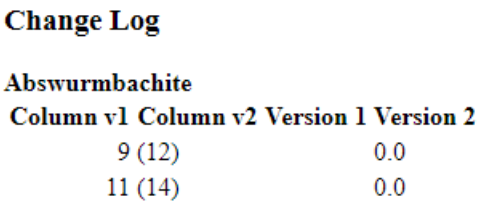
attribute to the Latitude and Longitude attributes, forming a single modification instead of an invalidate and two additions.

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

5.3 Produce Change Log

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

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



Change Log

Abswurmbachite

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

Figure 5.2: Abswurmbachite entry in the Copper Dataset Change Log

source from Figure 5.2 appears in Listing 5.1. Only the lines displaying table headers and the second data line have been removed. Immediately, it can be seen that the markup is quite cluttered as a result of the clash between RDFa's intent and its use. The goal of this technology is to describe the content on a page and give context to particular values, marking a string of digits as a phone number for example. However, in this application, very little of the content is used in the model except for line 3, which uses the mineral name as an attribute label. It is instead used to encode the versioning model into the HTML document's lattice, leveraging the ability to imply relationships in nested tags. Unfortunately, RDFa interpreters are very particular in the way they resolve implications, and the order in which data appears in a change log to be human readable does not match the order they need to be for RDFa to encode the model. In lines 10 and 11, two triples have to be explicitly included in order to conform with the model outlined in Chapter 4.

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

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

Listing 5.1: Abswurmbachite RDFa

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

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

```
     rdf-schema#label">Abswurmbachite</span>
  <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",
       "@id": "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/
          CUjsonlog.html#ModifyChangeAbswurmbachite12",
       "@type": "vo:ModifyChange",
       "resultsIn": "http://CUdb.com/v2/AttributeAbswurmbachite12"
},
{
       "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.
```

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

Listing 5.2: Abswurmbachite JSON-LD

5.4 Generate Versioning Graph

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

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

Listing 5.3: Noble Gas Add in Turtle

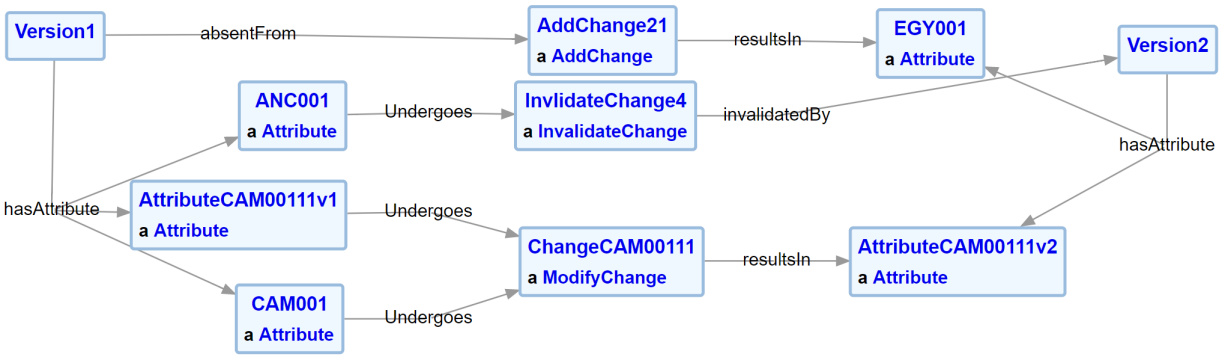


Figure 5.3: Some initial entries from versions 1 and 2 of the Noble Gas dataset

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

However, Figure 5.3 also demonstrates an interesting set of decisions made early in the generation process regarding modifications. Firstly, the relation presented in the figure is unbalanced and the right-hand side of ChangeCAM00111 links only to the column identifier but not to the corresponding row attribute. This links from a mismatch between the model's structure, the order in which data appears in the change log, and the way RDFa links properties together. Because the row label forms the outermost encapsulation, it cannot instantiate both row identifiers and implicitly link them separately. To do so would require explicitly instantiating the attribute in a non-visible part of the document which would defeat the purpose of using RDFa to implicitly encode the versioning graph into the document. Secondly, the column identifier AttributeCAM00111v1 combines together the row label and the column number, making it unique to just the CAM001 row. This was a decision

to properly identify the attribute as it appears in the change log document. Since the item is not a generic column 11 element in the log, but a specific item of CAM001, the entry id is included in the column identifier. This formulation poses a problem when trying to query the graph and determine how many rows have a column 11 modify change for example. When the versioning triples are not extracted from a change log and printed as linked data, the structure has greater freedom as seen in Figure 5.5.

5.5 Multiple Linked Versions

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

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

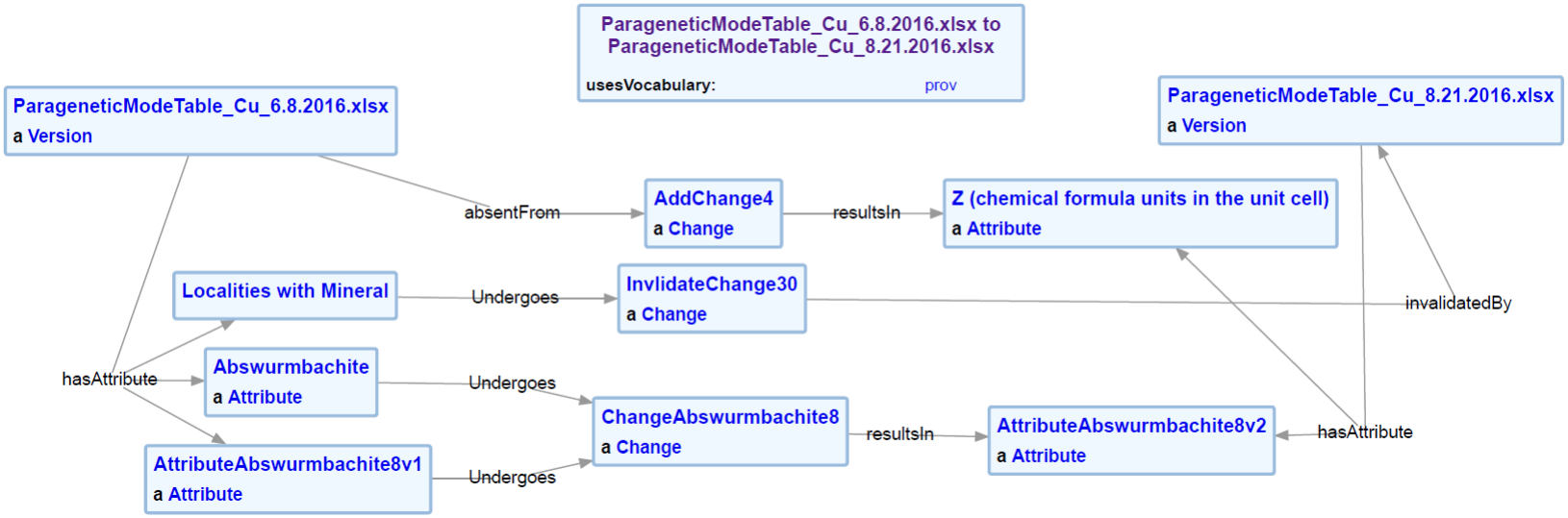


Figure 5.4: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications.

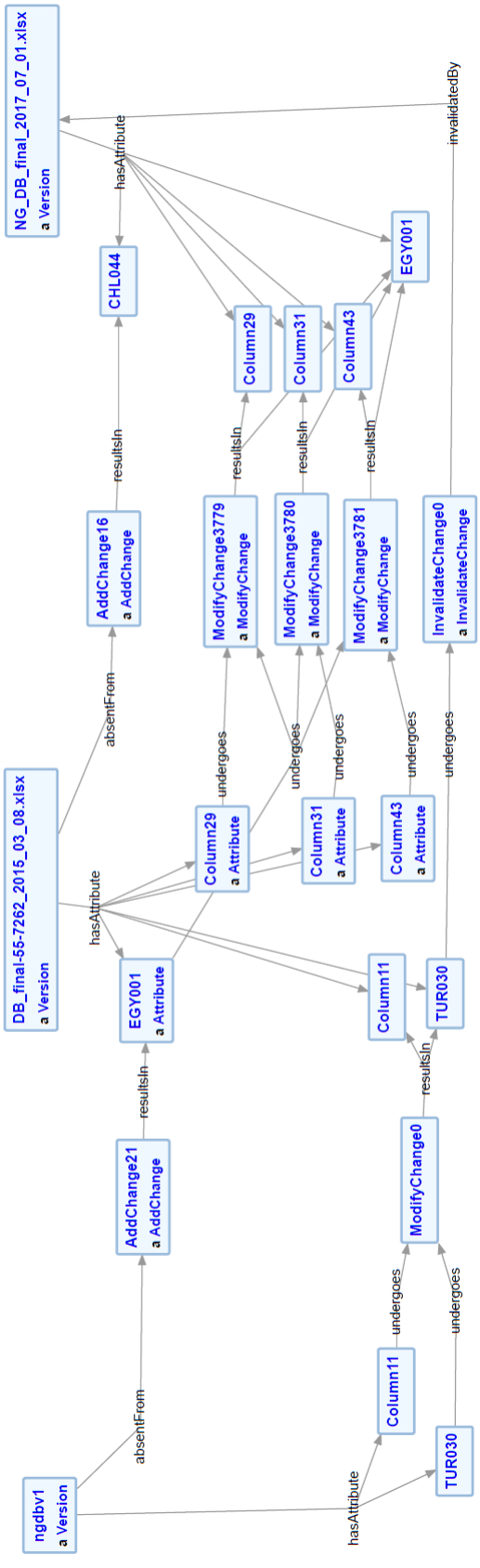


Figure 5.5: Versioning Graph representing the linked data graph with selected entries of additions, invalidations, and modifications after the publication of the third version.

CHAPTER 6
ONTOLOGY AND TAXONOMY VERSIONING

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

6.1 Creating the Versioning Graph

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

6.2 Quantifying Change

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

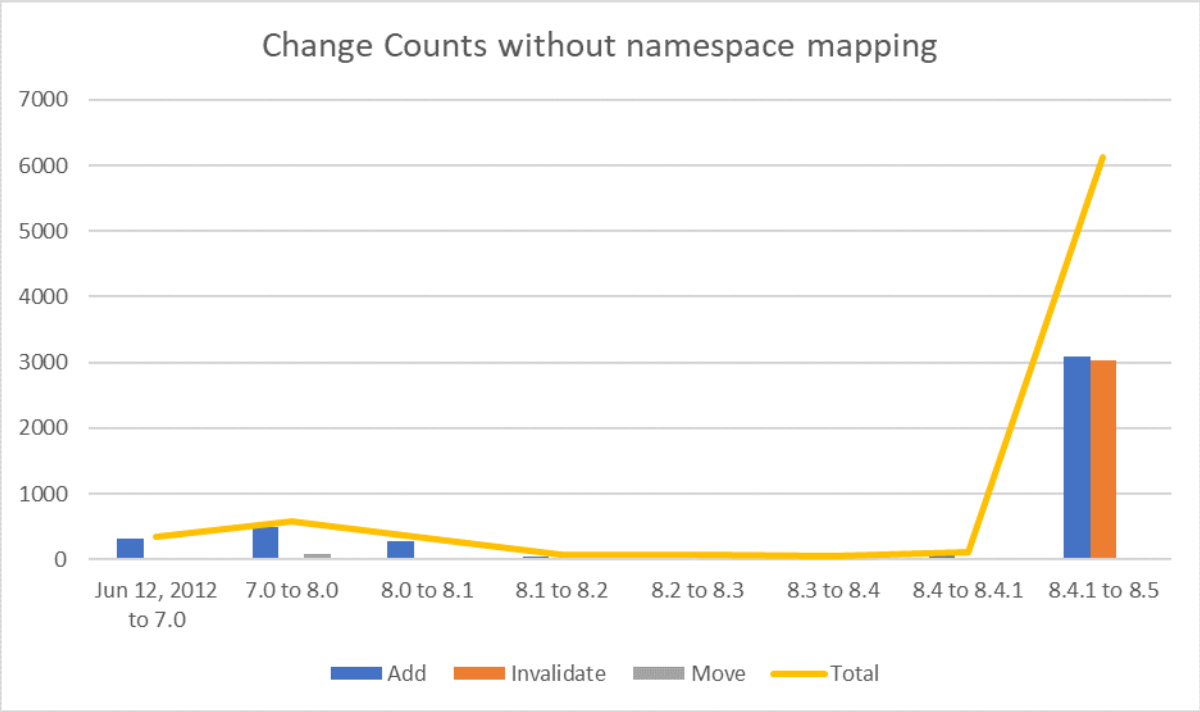


Figure 6.1: Add, Invalidate, and Modify counts in Version 8.5. The counts show change magnitudes and indicate that major and minor changes differ by orders of magnitude.

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

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

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

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

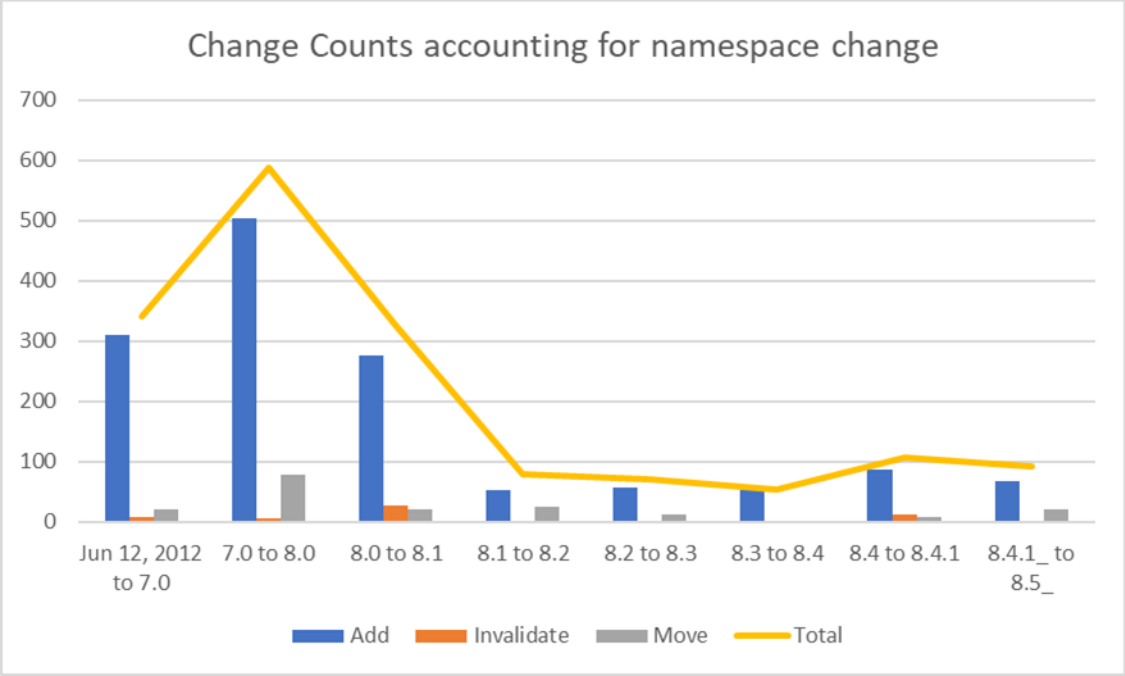


Figure 6.2: Add, Invalidate, and Modify counts ignoring the namespace changes in Version 8.5. The counts show change magnitudes appropriate for the identifier.

CHAPTER 7
MBVL CLASSIFICATION

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

7.1 Versioning Graph

The experiment undergoes two phases of comparison in this procedure. The first phase compares the initial species content with the classifications by a particular algorithm/taxonomy combination. Since the classification results from a population selected according to the initial species list, these two datasets share a common provenance. However, the list cannot be used in place of the classifier results because not only does it have only 21 entries, but also it does not have a clear correspondence with the DNA chains sent to the classifier. As a result, these two sets are not versions of each other, and versioning results will have weak implications. A labeling of the initial input data, could be considered a version of the results, but that data product is not available.

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

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

7.2 Analysis

Figure 7.1 shows the results of four comparisons performed using the matching procedure in the previous section. There are only four comparisons because varying both taxonomy and algorithm muddles the contribution of each towards a more accurate result. In the first set of columns, the Silva taxonomy results are versioned against RDP using the Spingo algorithm. The naming reflects the orientation in the versioning graph so Silva forms the left-hand version and RDP would be the right-hand version. In this comparison, using the RDP taxonomy seems to provide more accurate results, most specifically at the species level. The taxonomies also disagree fairly often at the species and family ranks. Switching to the Gast algorithm in the second set of columns, RDP once again demonstrates a noticeably greater accuracy in species classification. There 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.

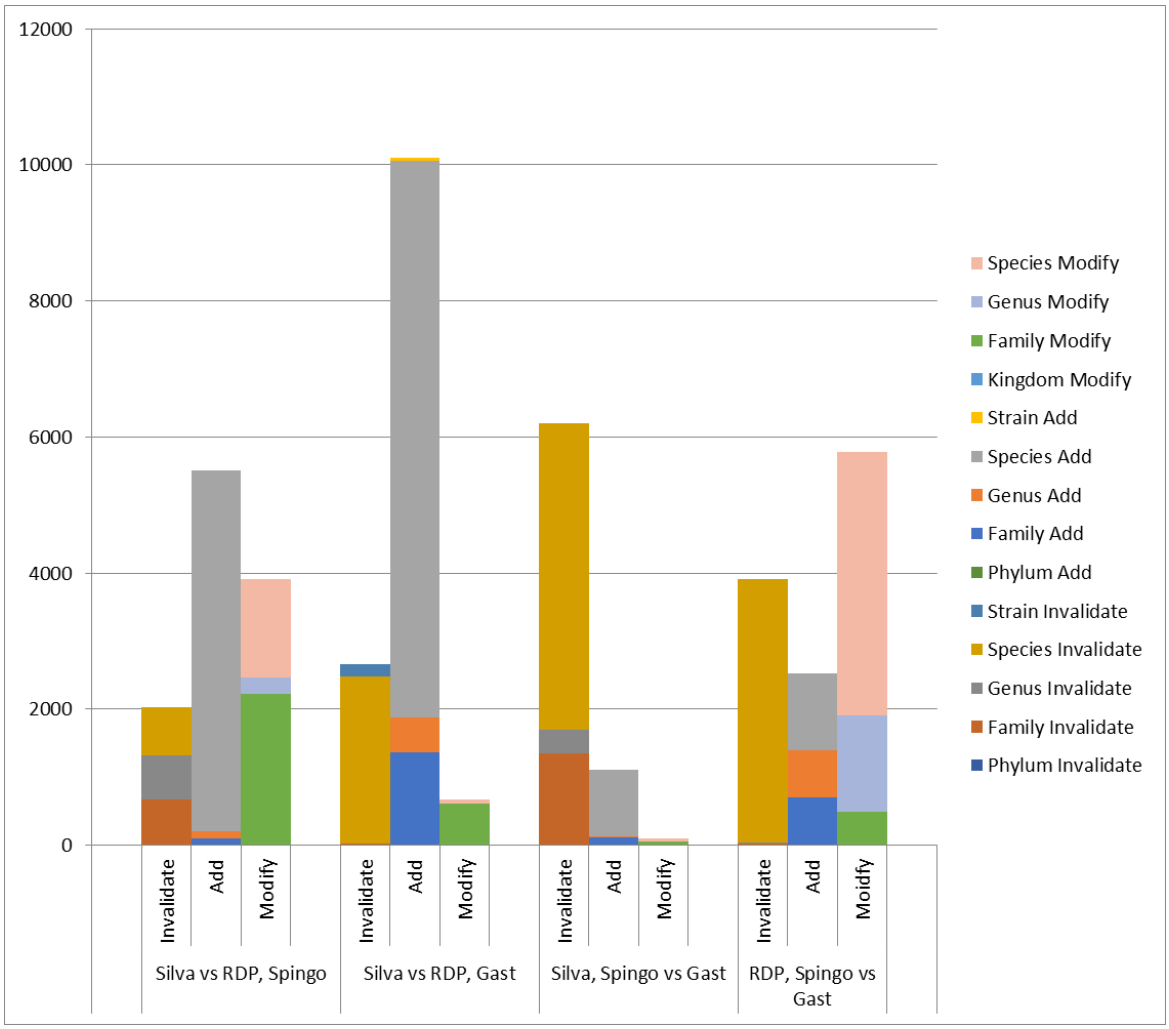


Figure 7.1: Compiled counts of adds, invalidates, and modifies grouped by taxonomic rank across algorithm and taxonomy combinations.

CHAPTER 8
DISCUSSION

8.1 Model

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

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

8.2 Implementation

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

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

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

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

8.3 Version Identification

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

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

8.4 Change Analysis

In Chapter **??**, the versioning process was used to compare the performance of different taxonomy and algorithm combinations. This diverges from many of the common understandings of derivations since each of the versions are not sequen-

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

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

CHAPTER 9
CONCLUSION

Formalizing versioning into a linked data model with a state-based approach and relating version change with changes in their attributes will improve the ability to standardly communicate version information. The proposed model does not possess the same brevity found in most popular provenance models, but it captures versioning as groups of relationships rather than sets of activities. This reveals a relationship between versions and their attributes in informing version changes. The approach led to difficulties viewing encoded versioning information in change logs due to excessive file sizes, but analysis suggests that there may be methods to merge multiple entries and improve performance. Future groups will need to perform further work and expand data change log adoptions and computability. The model also demonstrated that data distributors must find specific quantifiable measures to inform version identifiers since they play a key role in communicating information change. Direct edit counts, however, do not present a nuanced view of change impacts and should be viewed as initial characterizations of a version. Finally, we have demonstrated that versioning information can be used to perform services beyond documentation and tracking once a standard formal model has been produced. While often overlooked, version capture formalization will play a major role in data set evolution.

REFERENCES

[1] 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

[2] 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

[3] 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.

[4] 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

[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] 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

[8] 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

[9] 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

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

[11] "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

[12] 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

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

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

[15] 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

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

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

[18] 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.

[19] 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.

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

[21] 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/

[22] 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

[23] 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.

[24] 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

[25] 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.

[26] "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/

[27] "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/

[28] 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

[29] 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/

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

[31] 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

[32] 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

[33] 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.

[34] 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

[35] 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.

[36] 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

[37] 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

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

[39] 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

[40] 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

[41] 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

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

[43] 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

[44] 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

[45] 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.

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

[47] 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

[48] 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.

[49] 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.

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

[51] 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

[52] 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.

[53] 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.

[54] 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.

[55] 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/

[56] 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/

[57] 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/

[58] 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/

[59] 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/

[60] 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/

[61] 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/

[62] 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/

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

[64] 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/

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

[66] 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

[67] (2012, Jun.) Dcmi metadata terms. DCMI Usage Board. Accessed: February 8, 2017. [Online]. Available: http://dublincore.org/documents/2012/06/14/dcmi-terms/

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

[69] 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

[70] 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

[71] 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.

[72] 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

[73] 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

[74] 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.

[75] 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

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

[77] 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

[78] 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

[79] 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

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

[81] 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

[82] 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

[83] 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

[84] 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

[85] 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.

[86] 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

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

[88] 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/

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

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

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