Versioning Ontology for Encoding Change Log Data

BENNO LEE, Rensselaer Polytechnic Institute, USA
PETER FOX, Rensseleaer Polytechnic Institute, USA

Data often does not remain static after collection. Adding annotations, correcting errors, or removing invalidated data, a data set continues to evolve after collection. Current methods to track data change through linked-data ontologies utilize concepts from provenance ontologies, but a versioning ontology creates a more complete picture by focusing on versioning information. In this paper we define the Versioning Ontology (VersOn) with more complete semantics than current provenance ontologies. The VersOn then enables information within data change logs to be exposed as linked data using the Resource Description Framework in Attributes (RDFa) or JavaScript Object Notation for Linked Data (JSON-LD). We then evaluate the efficacy of VersOn by assessing the impact encoding VersOn into a change log has on the document's performance. Uncompressed provenance data is known to be on the order of the original data, forming an expectation of no more than 50% reduction in performance. VersOn encoded change logs show a much greater impact to performance.

CCS Concepts: • **Information systems → Semantic web description languages**; *Web log analysis*; *Resource Description Framework (RDF)*; *Web Ontology Language (OWL)*;

Additional Key Words and Phrases: Version, versioning, versioning ontology, data change, change log, provenance

1 INTRODUCTION

A change log is a document explaining the differences between two versions of a data object [6]. Valuable change information within the change log can be exposed as linked data using existing markup technology. A number of provenance ontologies include versioning concepts, but the terms define a specific view of change capture. Because provenance focuses on capturing the entities and activities used to produce a data object, provenance concepts do not completely capture the change differences between objects. Delving deeper and comparing the parts of the objects lies outside the scope of provenance information. In order to achieve completeness, a linked-data model must include the concepts of addition, invalidation, and modification. The model must relate the changing parts which differentiate objects as separate versions with the objects. The model must also maintain a continuity between versions to preserve the historical evolution of a data object.

In this paper, we establish the Versioning Ontology (VersOn), which satisfies the previous three completeness requirements, to expose information in change logs as linked data. Every change logged in the document costs a certain amount of storage space to store the change. One of the ways to measure change log performance is to compute the number of entries in the change log relative to the space necessary to store the information. We assess the capability of VersOn by looking at the reduction in performance due to adding the encoding markup compared to bounds formed from associated linked data.

2 PREVIOUS WORK

Versioning concepts in linked data can often be found in provenance ontologies such as The Provenance Ontology (PROV-O), a World Wide Web Consortium (W3C) recommendation [8]. PROV-O primarily focuses on the generation of data entities from activities, meaning the only entity comparison property is *prov:derivation*. Addition and invalidation properties result from data activities, improperly crediting the source of change from a versioning context. An addition or invalidation comes from the original or prior object, which may not be the object used by the data activity. The Provenance, Authorship, and Versioning (PAV) Ontology uses *pav:version* to label versions and *pav:previousVersion* to connect adjacent versions [7]. The ontology does not elucidate greater

Authors' addresses: Benno Lee, Rensselaer Polytechnic Institute, Troy, NY, 12180, USA, leeb5@rpi.edu; Peter Fox, Rensseleaer Polytechnic Institute, Troy, NY, 12180, USA, pfox@cs.rpi.edu.

detail explaining the differences between versions, identifying changing attributes or differentiating between additions, invalidation, and modifications. Schema.org is not an ontology but provides a means to identify different kinds of change with *schema:ReplaceAction*, *schema:AddAction*, and *schema:DeleteAction* [1–3]. The Schema.org concepts only model each change in isolation, not linking together a series of changes into a lineage. A data object is not always deleted when invalidated so *schema:DelteAction* is occasionally not appropriate to use.

Structured data allows the content of web documents encoded in HyperText Markup Language (HTML) to be understood as linked data. The Resource Description Framework in Attributes (RDFa) uses HTML tag attributes to annotate visible content, identifying a string of digits as a telephone number [10]. As will be seen in the later sections, the resulting model interacts very little with the visible content, making data storage more desirable than data annotation. JavaScript Object Notation for Linked Data (JSON-LD) enables structured data storage without involving visible content by introducing semantics to the JSON specification [11].

The amount of change information generated for an object can be related to provenance information since changes to activities and inputs result in new data objects which can be compared to the original object as a version, generating change information [4]. Buneman finds that provenance data consumes storage space on the order of the original data, or after compression, the provenance takes up twenty percent of the original space [5]. Including more data in a different form is guaranteed to decrease performance with the increased space utilization, but the reduction can be bounded. Since the change log contains the differences between two versions, therefore containing at most all the entries in the data set, encoding the change data into the log is expected to decrease performance by at most fifty percent. The storage space necessary to hold the original change log and the change data is double the original data set size in order to hold the original change data and the data in linked-data format in the same document, but the number of entries remain the same since the change log and change data entries describe the same changes.

3 THE VERSIONING ONTOLOGY

The model underlying The Versioning Ontology (VersOn) utilizes three primary concepts: version, attribute, and change. The model always relates at least two versions which represent the data object being versioned. An attribute is the part of a version which changes, causing a new version. A change represents the difference between two versions through their attributes, formed by the presence, absence, or alteration in a version. In order to represent all three kinds of change, addition, invalidation, and modification, the concepts are arranged and related in three different formations based on the presence or absence of attributes.

3.1 Modification

The modify relation occurs when an attribute appears in both versions and the attributes' value are different. In Figure 1, a modification is captured between two versions. Each version has an attribute, Attribute 1 and Attribute 2, respectively. A modification concept, a subclass of the change concept, connects the two attributes, denoting that the values described by the attribute differ. The connectivity between versions is maintained through the implied relationship across attributes. The attributes are directly associated with the respective version.

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

In some applications, a modification is represented as an invalidation followed by an addition. The representation has a couple of problems. The first is that the sequence of changes implies that there is an intermediary stage where all the modified values have been invalidated but not added. The intermediary stage constitutes a

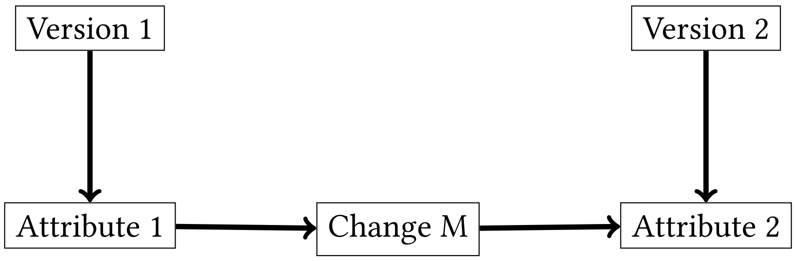


Fig. 1. Model of the relationships between Versions 1 and 2 when modifying Attribute 1 from Version 1 as a result of Change M, resulting in Attribute 2 from Version 2.

new version of the data object, even though only one change is being made. The second issue with using the two change sequence is that the two states of attribute, before and after the modification, becomes disassociated. Any new attribute being added to the version is not necessarily associated with an attribute in the left-handed version. Establishing the association between attributes after an invalidation-addition sequence is the same as and more concisely expressed as a single modification.

3.2 Addition

In Figure 2, the addition construction differs from the modify construction by the absence of Attribute 1. The absence creates a disconnect between Version 1 and Change A. In order to maintain version connectivity between versions, Version 1 must be reconnected to the other concepts in the model. A property is used to create a path from Version 1 to Change A, maintaining the connectivity between Version 1 and Version 2. The path does not show that Version 1 informs or creates Attribute 2, while the case may be true. The construction was also chosen to create a symmetric orientation with the invalidation change.

3.3 Invalidation

The invalidation construction has a missing attribute on the right-hand side of the relation, contrary to the addition construction. As a result of the invalidation, an attribute no longer exists in the right-hand version. As seen in Figure 3, the invalidation change concept matches to the Version 2 object. Just like in the addition model, the invalidation construction maintains a link between the two version objects in order to maintain version continuity. In the invalidation case, it makes more conceptual sense, however, because Version 2 invalidates Attribute 1 by omitting the attribute.

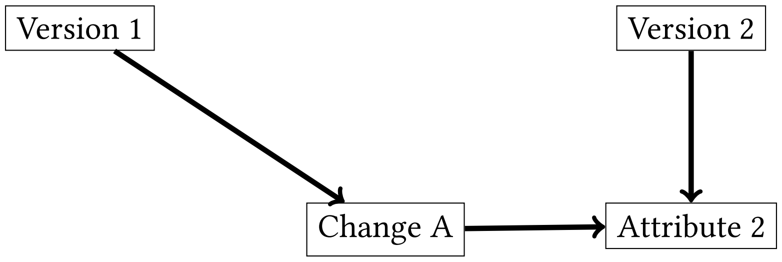


Fig. 2. Model of the relationships between Versions 1 and 2 when adding an Attribute 2 to Version 2 as a result of Change A.

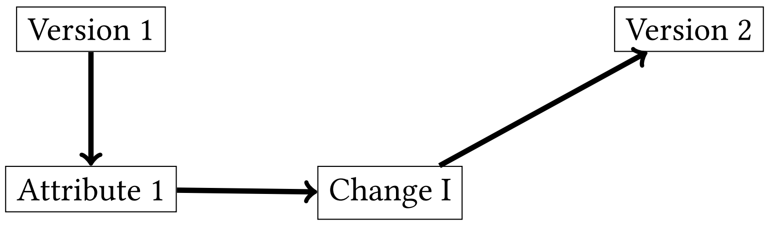


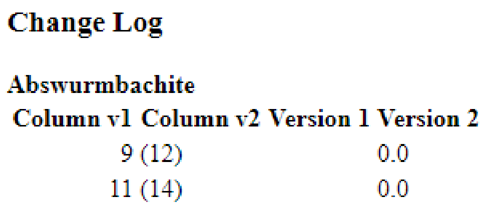
Fig. 3. Model of the relationships between Versions 1 and 2 when invalidating Attribute 1 from Version 1 as a result of Change I.

4 ENCODING A CHANGE LOG

The change log used in the following section was generated using data from The Paragenetic Mode for Copper Minerals database [9]. The change logs were marked up in HTML to facilitate RDFa or JSON-LD. Figure 4 shows the visible text of the mineral Abswurmbachite in the change log. Notice how the column indexes do not match across versions due to additions and invalidations of columns. The versioning model forms a conceptual mapping based on attributes, not on tabular indexes. The change logs follow a common format with three sections: Additions, Invalidations, and then Modifications. The sections may be further grouped by column or row additions. The division means that changes are not published into the change log as they are found, but instead organized and grouped beforehand.

4.1 Resource Description Framework in Attributes

Very little natural language is used in the change log to regularize the format and improve compatibility with RDFa. Listing 1 shows the text necessary to lay out the first four lines of Figure 4. While the content only shows four lines, the underlying markup takes up three and a half times as many lines. Line 2 of Listing 1 states that all following resources will be attributes of Version 1. Line 3 defines such an attribute. Lines 5 through 8 define the changes Abswurmbachite undergoes. Because RDFa embeds the statements within the content, the triples appear along with the described text. Lines 11 and 12 define complete triples which do not appear in the visible document. The lines complete the graph, but must be included in HTML span tags because RDFa only allows a single triple within each tag. Notice that the RDFa encoding does not interact with any of the visible content, but still relies on the tag structure to properly link concepts together. The resulting markup is completely constrained by RDFa's adherence to the visible content but does not leverage the benefits of RDFa's ability to incorporate the visible content.



Change Log

Abswurmbachite

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

Fig. 4. Abswurmbachite entry in the Copper Dataset change log.

```
1  <h3>Change Log</h3>
2    <div about="Version1" rel="vo:hasAttribute">
3      <div resource="v2:Abswurmbachite" typeof="vo:Attribute">
4        <span style="font-weight:bold"
          ↪  property="http://www.w3.org/2000/01/rdf-schema#label">Abswurmbachite</span>
5        <table rel="vo:Undergoes">
6          <tr  about="ChangeAbswurmbachite12" typeof="vo:Change">
7            <td align="right" rev="vo:Undergoes" resource="v1:AttributeAbswurmbachite12v1"
              ↪  typeof="vo:Attribute"> 9</td>
8            <td property="vo:resultsIn" resource="v2:AttributeAbswurmbachite12v2"
              ↪  typeof="vo:Attribute">(12)</td>
9            <td>          </td>
10           <td>       0.0</td>
11           <span about="Version1" property="vo:hasAttribute"
              ↪  resource="v1:AttributeAbswurmbachite12v1"></span>
12           <span about="Version2" property="vo:hasAttribute"
              ↪  resource="v2:AttributeAbswurmbachite12v2"></span>
13         </tr>
14       </table></div></div><br>
```

Listing 1. Abswurmbachite RDFa.

4.2 JavaScript Object Notation for Linked Data

After encountering the limitations of using RDFa to include the versioning ontology into the change log, JSON-LD was used. The JSON data is independent of the visible content's structure in the change log. Listing 2 provides the alternative encoding of the Abswurmbachite entry using JSON-LD. The entry is significantly longer, almost three times longer than the RDFa entry and ten times longer than the original visible content. Instead of including the data for each entry at the beginning or end of the document, each change block is separated into the particular *div* section for that change. This choice allows consumers to extract pertinent change information without needing to ingest the entire log.

The change logs created with RDFa or JSON-LD demonstrate progress towards documents which are both human and machine-readable. The implementation provides evidence that JSON-LD is better suited to embed a versioning graph into a change log than RDFa. RDFa suffers limitations since it is constrained by the content's structure. Notice the merged row and column identifiers in the `resource` attribute of lines 7 and 8 in Listing 1. The row identifier, Abswurmbachite, and column identifier, 12, were merged because the tag could only support a single value. The merged attribute stems from a mismatch between the model's structure, the order in which data appears in the change log, and the way RDFa links properties together. Because the row label forms the outermost encapsulation, data in the tag cannot instantiate both row identifiers and implicitly link the identifiers separately. To do so would require explicitly instantiating the attribute in a non-visible part of the document, defeating the purpose of using RDFa to implicitly encode the versioning graph into the document.

```
<h3>Change Log</h3>
  <div about="v1:Abswurmbachite">
```

```
    <span style="font-weight:bold"
    ↪  property="http://www.w3.org/2000/01/rdf-schema#label">Abswurmbachite</span>
    <table>
    <tr  id="ModifyChangeAbswurmbachite12">
      <td align="right"> 9</td>
      <td >(12)</td>
      <td>          </td>
      <td>       0.0</td>
      <script type="application/ld+json">
[
  {
    "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld",
    "@id": "http://CUdb.com/v1/AttributeAbswurmbachite9",
    "@reverse": {
      "hasAttribute": "Version1"
    },
    "@type": "vo:Attribute",
    "label": "Primary",
    "undergoes": "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/CUjsonlog.html#Modif⌋
    ↪  yChangeAbswurmbachite12"
  },
  {
    "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld",
    "@id": "http://orion.tw.rpi.edu/~blee/provdist/CU/DTDI/CUjsonlog.html#ModifyChang⌋
    ↪  eAbswurmbachite12",
    "@type": "vo:ModifyChange",
    "resultsIn": "http://CUdb.com/v2/AttributeAbswurmbachite12"
  },
  {
    "@context": "https://orion.tw.rpi.edu/~blee/provdist/GCMD/VO.jsonld",
    "@id": "http://CUdb.com/v2/AttributeAbswurmbachite12",
    "@reverse": {
      "hasAttribute": "Version2"
    },
    "@type": "vo:Attribute",
    "label": "Primary"
  }
]
      </script></tr></table></div><br>
```

Listing 2. Abswurmbachite JSON-LD.

Both structured data implementations break up the graph across attributes so that individual parts of the graph can be extracted. The practice of a one-node JSON object is generally helpful for many web applications to load data quickly, but since the change log is not an application, it makes more sense to break up the content. Changes to individual attributes can be identified using anchors on the web page, then agents need only extract and parse

Table 1. Copper change log size: 1st transition.

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

Table 2. Changes to Copper Data.

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

the linked data to the attributes' specific entries. This way, a subgraph of only the pertinent attributes can be created without first ingesting the entire versioning graph.

5 CHANGE LOG ANALYSIS

With a trade-off of 14 HTML lines for every visible line and 40 HTML code lines to each visible line, space utilization is a very present concern. Table 1 shows the size of each encoding of the change log as well as the percentage in size as compared to either of the files involved in the version transition. 'Text' denotes the encoding control where no structured data is included into the change log. The RDFa encoding is almost 6 times the size of the original files. The encoded change log exceeds the size of the control by more than ten fold, meaning over 90% reduction in change log performance. A separate file was generated in turtle format to observe whether taking just the linked-data values would reduce the information to a more manageable size, but the turtle file was still over 4 times the size of the original files. Adopting the versioning model and encoding it into a change log will very likely require significant storage investment.

Another way to evaluate the performance of the change log is to look at the number of change entries compared to the number of changed values in the Copper database's case spreadsheet cells. To determine the number of cells affected by a change, the number of cells added by new rows is summed with the number of cells added by new columns, using the width and length of Version 2. The cells affected by removals is based on the length and width of Version 1. The number of remaining equivalent values between the two files is 23940. Since modifications are reported cell-by-cell, the number of cells affected is equal to the number of modifications, 2628. The rows and columns that modification affects are not available because the changes appear inconsistently across the rows and columns meaning a reported value would be misleading. The complete counts are reported in Table 2.

The triples used to explain changes as a percentage of the cells affected are reported in Table 3. Smaller percentages indicate how well one triple can explain changes to multiple cells by compressing the number of entries. Notice that additions are much more efficient in explaining changes than invalidations due to a skew towards column additions. Invalidations explained changes to rows primarily while additions mostly explained changes to columns, but since columns are much longer, additions ended up scoring higher on efficiency. Modification triples do not compress change information well and also account for more than a majority of the changes to the data, meaning that modification triples most likely account for the bloat in the physical

Table 3. Change capture compression in Copper Data.

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

representation of the triples. Not represented in the change log are the unmodified cells which account for 89.02% of the matching cells between the Copper files. The analysis indicates that while addition and invalidation may be very efficient in expressing changes, improvements to encoding and modification capture are needed to bring down the storage costs of automated change logs.

6 CONCLUSION

Change capture using linked data is made more complete with the Versioning Ontology (VersOn). VersOn uniquely captures the three types of change, addition, invalidation, and modification, using three constructions. The ontology maintains connectivity between versions, preserving continuity across multiple versions. The model also correctly connects changing attributes with respective versions, not with provenance activities.

Initial encoded change logs were generated using RDFa. RDFa constrained the content and order of the encoding but could not be leveraged to utilize the visible data. An alternative encoding using JSON-LD pulled the linked data out of the attributes. The JSON-LD encoding can more closely adhere to the model at the cost of increased storage cost.

The automated change log generation yielded some unexpected results with reductions to performance of over 90%. While concessions were made using RDFa to keep the change log human readable, the linked data only log returned similar results. From the analysis, we can tell that significant summarization can occur if changes are summarized across rows or columns in a tabular data set. Because modifications are not summarized across rows or columns and comprise almost all of the changes in the encoding, modifications can be concluded to be the primary contributor to encoded change log size expansion.

REFERENCES

[1] [n. d.]. AddAction. http://schema.org/AddAction Accessed on: January 19, 2017.

[2] [n. d.]. DeleteAction. http://schema.org/DeleteAction Accessed on: January 19, 2017.

[3] [n. d.]. ReplaceAction. http://schema.org/ReplaceAction Accessed on: January 19, 2017.

[4] Bruce R. Barkstrom. 2003. *Data Product Configuration Management and Versioning in Large-Scale Production of Satellite Scientific Data.* 118–133. https://doi.org/10.1007/3-540-39195-9_9

[5] Peter Buneman, Adriane Chapman, and James Cheney. [n. d.]. Provenance Management in Curated Databases. In *Proc. of the 2006 ACM SIGMOD Int. Conf. on Manage. of Data.* 539–550. https://doi.org/10.1145/1142473.1142534

[6] Andrea Capiluppi, Patricia Lago, and Maurizio Morisio. 2003. Evidences in the evolution of OS projects through Changelog Analyses. In *Taking Stock of the Bazaar: Proc. of the 3rd Workshop on Open Source Softw. Eng.* http://roar.uel.ac.uk/1037/ Accessed on: July 12, 2014.

[7] Paolo Ciccarese, Stian Soiland-Reyes, Khalid Belhajjame, Alasdair JG Gray, Carole Goble, and Tim Clark. 2013. PAV ontology: provenance, authoring and versioning. *J. of Biomedical Semantics* 4, 1 (2013), 37. https://doi.org/10.1186/2041-1480-4-37

[8] Timothy Lebo, Deborah McGuinness, and Satya Sahoo. 2013. *PROV-O: The PROV Ontology.* W3C Recommendation. W3C. http://www.w3.org/TR/2013/REC-prov-o-20130430/ Accessed on: July 12, 2018.

[9] Shaunna Morrison, Robert Downs, Joshua Golden, Alex Pires, Peter Fox, Xiaogang Ma, Stephan Zednik, Ahmed Eleish, Anirudh Prabhu, Daniel Hummer, Chao Liu, Michael Meyer, Jolyon Ralph, Grethe Hystad, and Robert Hazen. 2016. Exploiting mineral data: applications to the diversity, distribution, and social networks of copper mineral. In *AGU Fall Meeting*.

[10] Manu Sporny, Ivan Herman, Ben Adida, and Mark Birbeck. 2015. *RDFa 1.1 Primer - Third Edition.* W3C Note. W3C. http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/ Accessed on: July 12, 2018.

[11] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindstrom. 2017. JSON-LD 1.1. https://json-ld.org/spec/latest/json-ld/ Accessed on: June 7, 2017.