[Demo] D2V: A Tool for Defining, Detecting and Visualizing Changes on the Data Web

Yannis Roussakis, Ioannis Chrysakis, Kostas Stefanidis, and Giorgos Flouris

Institute of Computer Science, FORTH, Heraklion, Greece
{rousakis,hrysakis,kstef,fgeo}@ics.forth.gr

Abstract. The dynamic nature of Web data gives rise to the need of understanding and analyzing the dynamics of individual datasets. As a matter of fact, the value of a dynamic dataset lies not only in its content, but also in its evolution history, which in some applications (e.g., trend analysis and identification), is more important than the data itself. In this paper, we present D2V, a tool that assists users in analyzing and visualizing the evolution history of dynamic RDF datasets. D2V is a flexible and powerful tool for defining, representing, detecting and presenting custom changes between datasets versions. We treat changes as first-class citizens, in a way that supports different modes of navigation on the deltas between any pair of versions, as well as the visualization of the evolution history of a given dataset.[1]

1 Understanding the Changes on the Data Web

Dynamicity is an indispensable part of the current data Web [1]; datasets are constantly evolving for several reasons, such as the inclusion of new experimental evidence or observations, or the correction of erroneous conceptualizations. Finding the differences (*deltas*) between datasets has been proved to play a crucial role in various curation tasks, such as the synchronization of autonomously developed dataset versions [2], or the visualization of the evolution history of a dataset [3]. Deltas are also necessary in certain applications that require access to previous versions of a dataset to support historical or cross-snapshot queries (e.g., [6]), to identify, for example, past states of the dataset, understand the evolution process, or detect the source of errors. In many cases, the information conveyed by the evolution history of a dataset is more important than the information in the data itself, e.g., for identifying trends or for understanding the evolution of a concept or an idea through time. Therefore, understanding, visualizing and analyzing the evolution history of a dataset is a critical component in applications dealing with dynamic datasets. Supporting this kind of analysis is based on four main pillars related to the *definition*, *representation*, *detection* and *visualization* of changes.

The first pillar amounts to identifying the changes appropriate for the application at hand, and their semantics. These changes will be the ones that the system should be able to understand and detect, and form the *language of changes*. Note that no language of changes is suitable for all applications, because different uses of the data may require a different set of changes being reported, or the definition of special changes

[1] This work was partially supported by the EU projects DIACHRON and IdeaGarden.

that happen often or are important for the operation of the data-driven application. To address this, we define two types of changes, aiming to make changes intuitive and human-understandable, while maintaining formal rigour: *simple changes* provide formal guarantees on the system's behaviour (i.e., satisfy the properties of *completeness* and *unambiguity* [4]) and are defined at design time, whereas *complex changes* are custom and defined at run-time by the user, in order to satisfy application-specific needs.

The representation of changes using an appropriate scheme is necessary to allow their persistent storage and support the subsequent visualization and analysis of the evolution history. Our approach is based on the definition of an *ontology of changes* and the representation of detected changes as instances of this ontology. This allows the connection of the detected changes with the actual data using standard linked data principles, blending the data with the evolution history and supporting navigation among versions, cross-snapshot or historic queries. The detection process allows the algorithmic identification of the deltas (based on the change definitions in the language of changes), as well as the storage of the detection results in the ontology of changes. To do this, we rely on the execution of appropriately defined SPARQL queries upon a given pair of versions; the answers to these queries determine the detected changes. The detection process is performed a posteriori, i.e., it does not require monitoring changes as they happen. This avoids the need to have a controlled system, and allows remote users of a dataset to identify changes, even if they have no access to the actual change process.

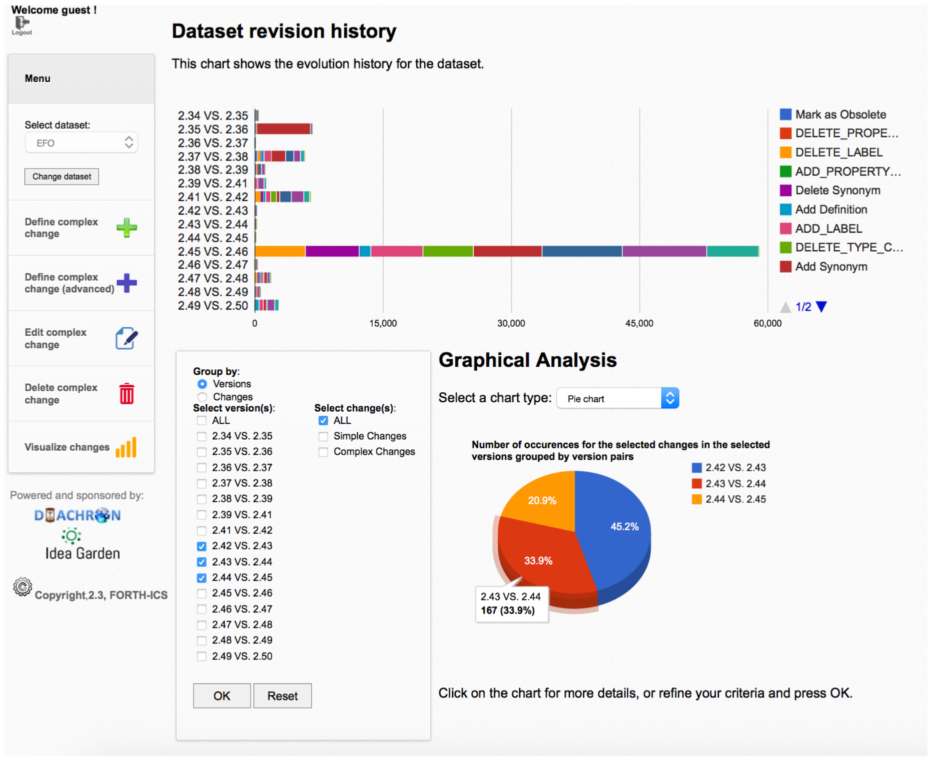
Eventually, the retrieval of the detected changes allows their presentation to the user through appropriate interactive interfaces and visualization paradigms. As changes are stored as RDF data, our approach relies on SPARQL queries for retrieving the evolution information. We provide different views of the evolution history to allow different types of analyses: for example, the user may want to see the evolution history of a given URI (*term-centric view*), of the dataset as a whole (*dataset-centric view*), or of specific versions (*version-centric view*); or he may be interested in a *change-centric view*, where the instantiations of a given change are reported; he may want to filter the different results to a fixed set of changes, change types, or versions; or he may want to visualize evolution along a series of consecutive versions, or for an arbitrary pair only.

To support these needs, we developed a flexible tool for change detection and analysis of RDF datasets. This tool is based on a theoretically solid and generic methodology for addressing the different problems related to change analysis, which is described in [5]. This paper focuses on describing the tool, which can be found at `http://www.ics.forth.gr/isl/D2VSystem`; a video showing its main features can be found at `http://www.ics.forth.gr/isl/D2VDemoVideo`.

2 Visualizing the Changes on the Data Web

D2V provides functionalities for managing complex changes, namely defining, editing and deleting them; this management refers to a specific dataset (chosen from a drop down menu in the upper-left corner of the interface), as different datasets may allow for a different set of complex changes (Figure 1-left).

We provide two modes for change definition, which allow for a different range of changes to be defined, but also require a different level of familiarity of the user with the


Welcome guest !

Dataset revision history

This chart shows the evolution history for the dataset.

Select dataset: EFO

Graphical Analysis

Select a chart type: Pie chart

Group by: [✓ Versions] Changes

Select version(s): 2.42 VS. 2.43 ✓, 2.43 VS. 2.44 ✓, 2.44 VS. 2.45 ✓

Select change(s): ALL ✓

Number of occurences for the selected changes in the selected versions grouped by version pairs

2.43 VS. 2.44
167 (33.9%)

Click on the chart for more details, or refine your criteria and press OK.

Powered and sponsored by:

Copyright,2.3, FORTH-ICS

Fig. 1. Dataset-centric and version-centric view.

system. The first mode is meant for less experienced users. It allows the definition of a complex change with minimal input, according to certain templates that correspond to interesting changes. More experienced users could use the advanced definition mode, where many different options are allowed, and the range of possible changes to be defined is much larger; on the down side, this interface requires some familiarity with the semantics of the simple changes. Note that the definition, edit and deletion of complex changes might affect the already stored information on previous detections; therefore, after each definition, edit and deletion, the system will have to partly redo some of the change detections to update the already stored changes.

The management of complex changes can be done under the three supported log-in options: a shared (guest) account, a personal account with predefined complex changes, and a blank personal account. Personal accounts can be used to create a personal space of complex changes to define changes that are useful for a certain user or application, and guarantees no interference from other users; the guest account provides a shared “sandbox” for experimentation. To help first-time users, personal accounts can come as totally blank (no changes defined) or with some predefined changes for guidance.

The visualization interface is at the core of our demo, and it allows the analysis of the evolution history of a dataset in general, as well as the comparison of any two arbitrarily selected dataset versions. The operation mode is selected in the opening option menu of visualization. We will first consider the case of evolution history analysis, where the user is first presented with the *dataset-centric view*, showing the entire evolution history along all ingested versions of the chosen dataset. This allows viewing all the changes that were detected between each pair of consequent versions – hovering over the bars shows additional statistics (see Figure 1-top, for the EFO dataset[2]).

Below the dataset-centric view, the user is presented with a panel providing a wide variety of different visualization options, allowing more customized analyses. This interface allows a *version-centric* or *change-centric* view, where changes are grouped by version pair, or by change type; the results can be filtered for specific version pairs or change types, and grouped accordingly (Figure 1-bottom). As in the dataset-centric view, the numbers are relative, but hovering over the graphic gives absolute numbers. The chart type can be changed, allowing pie, column, bar, line and area charts to be used, as well as a tabular form that can be sorted in descending or ascending order by clicking on a column name. These charts are useful for understanding the mix of changes in each/across version pair(s), as well as the number of appearances for each change across specific version pairs. For more details, a click on a chart element (e.g., on a piece of the pie), will show analytical details about the changes being visualized in this chart element (namely, all changes appearing in the corresponding version pair, or all detected instantiations of the corresponding change in the selected version pairs).

Further, clicking on a term in the latter table, returns another table presenting all the different change instantiations involving this term, turning the view into *term-centric*; this is useful to understand the evolution of a term over time. The results in this table are filtered according to the chosen change types and versions. If the operation mode is changed to the comparison of a custom pair of versions, the interface and functionality is similar; the main difference is that now the comparison is restricted to the version pair selected by the user in the opening screen. The user can select any pair, even versions that are not consecutive. Another difference in this scenario is that the changes are not pre-detected, so the system will have to call the change detection algorithm at run-time.

References

1. V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Synthesis Lectures on The Semantic Web. Morgan & Claypool Publishers, 2015.
2. R. Cloran and B. Irvin. Transmitting RDF graph deltas for a cheaper semantic Web. In *SATNAC*, 2005.
3. N. F. Noy, A. Chugh, W. Liu, and M. A. Musen. A framework for ontology evolution in collaborative environments. In *ISWC*, 2006.
4. V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, and V. Christophides. High-level change detection in RDF(S) KBs. *ACM Trans. Database Syst.*, 38(1), 2013.
5. Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris, and Y. Stavrakas. A flexible framework for understanding the dynamics of evolving RDF datasets. In *ISWC*, 2015.
6. K. Stefanidis, I. Chrysakis, and G. Flouris. On designing archiving policies for evolving RDF datasets on the Web. In *ER*, 2014.

[2] `http://www.ebi.ac.uk/efo/`