# Visualization for Ontology Evolution

Patrick Lambrix[1], Zlatan Dragisic[1], Valentina Ivanova[1], and Craig Anslow[2]

[1]Department of Computer and Information Science
and the Swedish e-Science Research Centre, Linköping University, Sweden
[2]Department of Computer Science, Middlesex University, London, UK

**Abstract.** One of the challenges for the ontology engineering community is the user involvement in the engineering process. Ontologies are not static entities and there is a demand for tools to support the user during the ontology evolution process. This paper aims to provide a set of functionality requirements for ontology evolution systems, with a particular focus on the visualization of the ontologies, their versions and information needed for ontology evolution tasks. Further, we review the current state of the art in ontology evolution systems with respect to the requirements and the visualization. We also view ontologies as software and discuss approaches from the software visualization area that could be used for ontology evolution visualization.

**Keywords:** ontology evolution, ontology visualization, software visualization

## 1 Introduction

Ontologies are a key technology for the semantic web and are used in semantically-enabled applications. Ontologies are not static entities but evolve over time. In [36] ontologies are classified into initial (with a large number of changes), expanding (many additions, with deletions and modifications), refining (mainly refining existing concepts), mature (some additions and modifications, few deletions) and dormant (little activity). There are different reasons for changes in ontologies. In [34], a study on the evolution of Gene Ontology, the authors identified the following reasons for change: (1) dealing with anomalies (essentially modeling defects), (2) extending the scope to take into account new fields, (3) dealing with diverging terminology across communities (e.g., use of the same name for similar but not exactly the same processes in plants and animals), (4) mirroring scientific advance and (5) adding relations between ontology terms. Several other papers report the fourth reason that includes new discoveries and changes in the domain, e.g., [11, 44, 54]. The first reason can be extended in scope to also include semantic defects. These defects could be found by inspection by domain experts or as a result of a debugging and completion step in the ontology development process [15] using tools as RepOSE [24, 30] or OOPS! [45]. Another reason for change discussed by [44, 54] is the need to match the changing activities of the users.

In practice ontologies are used in a number of semantically-enabled applications in which the evolution plays an important part (Case 1). One of the ways to use ontologies for *semantic search* is query expansion, where, based on ontologies, an original query is expanded to take into account synonyms of the terms in the query, more specific terms, or other related terms. When the ontologies change, the results of the queries may change as well [15]. Ontologies are also often used for *data integration* where the ontologies take the role of content explicitation, e.g., [49]. For the integrated data sources the ontologies can also take the role of query model and during the integration the ontologies can be used for verification. When the ontologies change, the data sources may not be correctly integrated anymore. Further, many data sources, e.g., in the biomedical domain, annotate their entries with ontology terms [28]. This allows for browsing a data source using an ontology, helps with semantic search as well as with data integration. Annotations can also be used in specialized data analysis such as functional enrichment analysis. When the ontologies change, the semantics of the annotations may also have changed and therefore not be complete or appropriate anymore. For instance, [19] reports on the impact of the evolution of Gene Ontology on functional analyses.

Ontology evolution is also used for obtaining knowledge about the evolving ontology (Case 2). Information about the evolution can be used for *quality assessment* for the ontology. For instance, in Evolutionary Terminology Auditing [8] adding terms to a new version of a terminology reflects dealing with unjustified absences while deleting terms reflects dealing with unjustified presences in previous versions of the ontology. In [36] the evolution of ontologies is used to classify the ontologies into five profiles of *activity*: initial, expanding, refining, mature and dormant. Further, information about how the ontology evolves allows us to find *trends*, what changes are dominating and what has been changed and what not, the latter which is particularly important in collaborative ontology development [21].

Although ontology evolution is considered important in ontology engineering (and should be part of mature ontology development tools), there are few tools. Further, one of the current challenges for the ontology engineering community is the user involvement in the engineering process. Although ontology development tools such as Protégé have large support for user interaction, it is not always the case for systems focusing on other ontology engineering tasks. As an example, in the ontology alignment task, that produces mappings between ontologies, it has been recognized that user involvement is necessary in the validation phase, but the performance and quality of the final set of mappings could also be significantly improved with user involvement in the algorithm selection and mapping generation [46]. Therefore, requirements for such systems and their user interfaces have been proposed [17, 25] and some systems are focusing on user involvement [29].

In this paper we discuss how user involvement can and should be introduced in tools supporting an ontology evolution process. After introducing an ontology evolution methodology and discussing the types of changes in ontologies in Section 2, we introduce desired functionality for ontology evolution systems in Section 3. In Section 4 we provide a literature review of how current systems support the desired functionality with a focus on visualization. As ontologies are also software we also look into the software evolution area and review the used visualization techniques in Section 5. We conclude the paper with a short discussion in Section 6.

## 2 Ontology Evolution

### 2.1 Ontology Evolution Process

Several methodologies for dealing with ontology evolution have been proposed. We briefly describe the process recently proposed in [54], that extends or overlaps with several previously proposed methodologies, e.g., [13, 18, 47]. The proposed process for ontology evolution contains five steps. The first step deals with detecting the need for evolution. This need could be based on any of the reasons for change described in Section 1 and can be initiated by domain experts or through analysis of data directly related to the ontology (e.g., structure, instances) or indirectly related external sources (e.g., text documents, databases, queries to data sources annotated with the ontology's terms). In the second step changes are suggested. The suggested changes can be generated by extracting entities from text documents, by using ontology learning techniques, using lexical databases as WordNet, using other ontologies as background knowledge or using debugging and completion systems. The changes suggested in the second step are validated in the third step. The suggested changes are validated regarding their correctness in the domain as well as regarding logical properties. In the latter case it is checked that the changes do not introduce inconsistency or incoherence. In the fourth step the impact of the changes to external artifacts is assessed. This includes invalidation of data instances, dependent ontologies and applications. Finally, in the fifth step the changes are managed which includes keeping track of the changes and the different versions of the ontology. In the case that direct recording of changes is possible, a change language is needed. If changes cannot be recorded, approaches for change detection should be used. We refer to [54] for an overview of tools that can be used for the different steps.

We note that the process does not explicitly take into account case 2 where the evolution is used to gain information about the ontology itself (quality, trends), although one could argue that the information gained from case 2 may be used to decide on the need of changes (step 1) or for suggesting changes (step 2).

### 2.2 Types of Changes

Different authors classify the changes in different ways. In [44] different terms are used for the ontology evolution based on the kind of change. Ontology extension is used when new single elements are added. Ontology refinement is used when there is an addition of new concepts where an is-a relation is established between an existing concept and a new concept. Finally, ontology enrichment encompasses the addition of non-taxonomic relations or other axioms. In [27] a distinction is made between non-logical changes (e.g., change in the natural language description of a concept) and logical definition changes which affect the formal semantics.

Most papers identify the addition and deletion of concepts, relations, instances and axioms as single or elementary changes, e.g., [23, 27, 47, 48]. Some approaches consider modifications as elementary changes while others consider modification as a combination of deleting and adding. When introducing changes, the user may be interested in a higher-level change, e.g., substituting a concept by another concept, rather than in the actual sequence of elementary changes that implement the higher-level change.

Therefore, some approaches also introduce composite or complex changes. For instance, [21] introduces mapping concepts in different ontologies, substituting concepts, moving a concept and its descendants to another place in the is-a hierarchy, changing attribute values, merging concepts, splitting concepts, adding, deleting, merging and splitting for leaf concepts, adding and deleting subgraphs, and making a concept obsolete or revoke this decision. In [47] variants of merging, splitting, and moving as well as copying are introduced. In [48] complex changes are user-defined.

A study on how users edit ontologies [50] proposes that the hierarchical structure of the ontologies has the strongest influence on the editing behavior. Other influencers are the entity similarity and the semantic distance of concepts. Further, users edit the ontology in a combined top-down and breadth-first fashion.

## 3 Functionality for Ontology Evolution Systems

There are a number of tasks that need to be performed in cases 1 and 2 which lead to desired functionality for ontology evolution systems, and to be able to provide this functionality different kinds of information needs to be gathered and analyzed.

The approaches that we reviewed start with different versions of an ontology or support creating new versions. However, in [36] it was observed that discovering previous versions of ontologies is not always straightforward. For many ontologies there are issues with provenance and we may not find all versions.

Based on a study of projects that use Protégé (Perot Systems, NCI, OBO), it was stated in [39] that the following features are often requested.[1] Users requested the change history of a concept as well as the representation of changes between ontology versions. This should include information about the actual change as well as other provenance information such as who edited, when and why. (These features can be used in steps 1-4.) To be able to provide provenance information change annotations should be supported (step 5). In the case changes are not recorded, functionality to compare different versions should be provided (steps 1-4). A printed as well as electronic summary of changes between versions should be available (steps 1-4). Also a specialized view of changes (e.g., the changes by a specific author) was requested (steps 1-4). Users also wanted to be able to query old versions of the ontology using terminology of the new version (steps 1-4). Mechanisms for the identification of conflicts as well as to accept and reject changes should be provided (step 3). Additional features include a roll-back mechanism, the ability to save the current state in the middle of the reviewing of an ontology, and the ability to post and describe new versions. As the study included collaborative ontology development, the users also requested access privileges such that work of different authors would not clash. Further, there was a need for a negotiation mechanism to resolve conflicts that occurred as a result of work by different authors.

The functionality to compare different versions is also requested by other authors, not only when the changes cannot be recorded, but also as a way to discover trends and gaining insights regarding change and stability for the ontology, e.g., [5, 9]. Additional

[1] We annotate in parentheses the features with the number of the step in the process in Section 2.1. We also note that many of the features can be found in other articles.

**Table 1.** Functionality for ontology evolution systems.

| Step | Category | Functionality |
|---|---|---|
| step 1 | I | 1 show an ontology version |
| | I, E | 2 show different ontology versions in evolution graph |
| | I, E | 3 show changes/diff between ontology versions (with change types) |
| | I | 4 show summary of changes |
| | I | 5 show specialized view of changes |
| | I | 6 show change history of a concept/relation |
| | I, E | 7 show provenance information |
| | I | 8 show information about/context of concept/relation |
| | I, E | 9 compare different versions |
| | I | 10 search and query ontology |
| | I | 11 query old versions using terminology of new version |
| | I, E | 12 discover trends |
| | I, E | 13 discover volatile and stable regions |
| step 2: | - | 1 identify and suggest changes |
| step 3: | - | 1 identify conflicts |
| | I | 2 show conflicts |
| | M | 3 resolve conflicts |
| | M | 4 accept and reject suggested changes |
| step 4: | - | 1 evaluate influence on dependent artifacts |
| | I | 2 show influence on dependent artifacts |
| | M | 3 update of dependent artifacts |
| step 5: | M | 1 execute changes |
| | E | 2 identify and show implications of change in ontology |
| | M | 3 add/edit change annotations |
| | M | 4 roll-back mechanism |
| | - | 5 save current state |

functionality that can be found in the literature includes detecting the changes, evaluating the influence of the evolution on dependent artifacts and semi-automatic updates of these artifacts to reflect evolution, e.g. [11].

We show a summary and, based on our experience in ontology engineering, a slight extension, of the functionality in Table 1. We have added the functionality for case 2 as part of step 1. For the functionality that presents information or requires interaction we also categorize the functionality in terms of inspection (I, e.g., exploring, searching), manipulation (M, adding/transforming information) and explanation (E) as in [25].

Different kinds of information are needed for the requested functionality (Table 2). For most kinds of functionality we require information about the actual ontology versions and their concepts, relations, axioms and instances. Some functionality such as discovering trends or volatile and stable regions [12, 23] needs statistics about the ontology versions. Similarly, most of the functionality requires information about the changes [8, 27], while functionality such as discovering volatile and stable regions also requires statistics about the changes [23, 36]. Provenance information [47, 8] is used in decision making related to several kinds of functionality, e.g., resolving conflicts and

**Table 2.** Information for ontology evolution functionality.

| **Category** | **Components** |
|---|---|
| Ontology and ontology versions representations | concepts, relations, instances, axioms [41, 43, 47] |
| Ontology version statistics | number of concepts/relations/instances [22, 20],<br>number of structural relations (is-a, part-of) [22, 20],<br>concept types (obsolete vs non-obsolete; leaf vs inner node) [22],<br>proportion of leaves [12],<br>nodes' average height and average depth [12],<br>in- and out-degrees of nodes [12, 23],<br>number of paths, path lengths [23] |
| Change representations | logs of elementary and complex changes [41, 43, 47, 20] |
| Change statistics | number of concepts and relations added/deleted [22, 20, 9, 48],<br>number of axiomatic changes [22, 20],<br>changes wrt time interval [22, 9],<br>add-delete ratio, growth rates [22, 20, 9] |
| Provenance | change author [41, 43, 47], change time [41, 43, 47, 22, 9],<br>cost of change [47, 9], cause of change [47],<br>change description [41, 43, 47, 22, 9] |
| Connections | ontology version level connections [27],<br>conceptual relations between concepts/relations in different versions [20, 27] |

executing changes, as well as for computing specialized views of changes. To be able to show the ontology evolution we need information about the connections between different versions of the ontology as well as the conceptual relations, e.g., equivalence and is-a, between the concepts and relations in different versions [27].

## 4 Ontology Evolution Visualization

We conducted a literature review of current ontology evolution systems and discuss how they support the desired functionality as in Table 1 with a focus on visualization[2].

CODEX [20] can be used in parts of cases 1 (for semantically-enabled applications) and 2 (obtaining knowledge about the evolving ontology). It provides support for determining complex changes between two versions of ontologies (1.3). It uses COnto-Diff [21], a rule-based algorithm for computing the complex changes. The user interface is composed of multiple views. A high-level view provides statistics about the number of relations and concepts in two ontology versions as well as the number of changes between the versions, both simple and complex changes (1.4). The distribution of different change types is presented in the form of a piechart. The change explorer view and the change navigator view provide support for navigating through changes from

[2] We denote the functionality as x.y where x is the step number and y the functionality number within the step.

the complex ones to simple ones. The change explorer view utilizes tag clouds to show the frequency of different change types or number of times a concept has been changed (1.12). After selecting a change in both explorer view and change navigator view the changes can be explored in a tree-like manner. Finally, the change impact view gives a possibility of exploring which of the provided list of items were influenced by a change (5.2). Again, changes are explored in a tree-like manner.

REX [9] is tool for case 2, focusing on exploring evolution of ontology regions. A region is defined as an ontology concept with its associated is-a subgraph. The extent of changes in a certain region of an ontology is defined via a cost model which assigns costs to ontological changes. The model is customizable. The tool consists of three components: structural analysis, quantitative change analysis (1.4) and trend analysis (1.3, 1.6, 1.9, 1.12-13). The structural analysis component provides two views, a table view and a browser view. The table view is in the form of a spreadsheet containing information about the average cost of a region as well as concepts contained in the particular region. Average cost is defined as total cost of changes in a region divided by the size of the region. The browser view is in the form of a fisheye view of an ontology version. The ontology version is represented as a graph where nodes represent concepts and edges represent is-a relations. The fisheye view implies that the selected concept is in the center while its subconcepts are organized around it. The color of a node describes the concept's change intensity, red implies high change intensity (volatile region), while green marks stable concepts (stable regions). The quantitative change analysis component provides information on how many changes of certain type occurred in a specific time interval. The information is shown in the form of a line chart with change count on one axis and ontology versions (time) on the other axis. The trend analysis component provides a means for studying and comparing evolution of regions. Users can select the time interval as well as regions of interest. A line chart shows the average cost for selected regions at different time points. Partial provenance information (time of changes) is provided by the tool (1.7).

OnEX [22] can be used in parts of case 1 (by migrating annotations) and case 2. It is a tool for exploring ontology changes (1.3, 1.9) and implements two ways for following an ontology's evolution: quantitative evolution analysis and concept-based analysis. The quantitative evolution analysis gives an overview of an ontology's evolution (1.4) in the form of spreadsheets with different levels of granularity. For example, the high-level view provides the number of concepts and relations in the first and current version of all ontologies in the repository. A chart gives an overview of the trends in the number of relations and concepts over time for a selected ontology (1.12). After selecting an ontology the user is presented with numbers for different change types between versions of the ontology. Selecting a change type for a version gives a list of changes of this particular type that were done in this version. In the concept-based analysis the users can follow a concept's evolution (1.6). A concept's evolution is presented in a spreadsheet with information such as the date of change, old and new values, type of change (1.7).

PromptDiff [41] is a Protégé plugin for ontology evolution. It covers both cases. The framework for ontology evolution consists of two components: a change management plugin and a plugin for comparing ontology versions. The change management plugin provides a list of changes made to an ontology with associated annotations (time, author, comment) (1.7, 5.3). It also provides two views for changes. The detailed view provides information on individual simple changes, while the summary view groups together simple changes. The plugin for comparing ontology versions (1.3, 1.9) visualizes indented trees of two ontology versions as a single tree. It allows visualization of concept level changes as well as tree-level changes. Color coding and symbols are used to visualize changes. For example, names of new concepts are underlined, names of deleted concepts are crossed out, names of moved concepts are greyed out in the old position and bold in the new position. Visualization of ontology versions (1.1) and editing/searching support is provided directly by Protégé (1.10, 5.1, 5.3-5).

OntoView [27] covers parts of cases 1 and 2. It is a system which supports users in specifying relations between ontology versions (1.3, 1.9). The tool also provides some limited support for analyzing effects of changes (5.2). It highlights the places in the ontology where changed concepts and relations are used. The visualization for comparing two ontologies is provided in the form of a unix-style diff, i.e., ontologies are given side by side (in XML format) and changed parts are highlighted. The user can characterize the conceptual implications of a change using a menu as identical (no change in meaning, only explanation is changed) or conceptual (in this case a relation between the two versions of a term can be given as e.g., a subsumption relation).

The NeOn toolkit [43] focuses on case 1 and provides support for the ontology evolution process via different plugins. The RaDON plugin is used for diagnosis and repair. The Evolva plugin is used for discovery of changes from external data sources and checking the relevance of a change. The toolkit provides limited decision support for accepting/rejecting changes by presenting a list of side-effects for a change (5.2). A change capture plugin allows for logging of changes with provenance information such as author, time, type of change, etc. (1.7, 5.3). The logs are presented in the form of spreadsheets. Ontology versions are visualized as trees (1.1), however there exist other plugins for visualizing ontology versions.

The KAON Framework [47] is used in case 1. The user has to define an ontology evolution strategy which defines how to deal with consequences of a change (e.g., what to do with instances of a deleted concept). After applying a change, the system computes consequences of a change given the defined strategy and presents it to the user in a spreadsheet. In the KAON framework ontology versions are visualized as graphs (1.1). The framework provides support for executing changes (5.1), querying and searching (1.10), undo/redo (5.4) as well as saving the current state (5.5). Identifying and suggesting changes is implemented via a companion tool Text2Onto.

The Ontology Lookup Service provides possibilities to visualize the evolution of ontologies [48]. It performs change detection and visualizes the number of different types of simple changes (1.3-4). By clicking on the numbers the user can retrieve information about the actual changes. Ontologies can be shown as indented trees (1.1) and queried (1.10).

Although not directly related to any system, in [5] visualizing ontology evolution with dynamic graphs [4] is discussed. They distinguish time-to-time mappings which are animations, and time-to-space mappings which are static displays. As these are graph visualizations they need to take into account requirements regarding the representation of nodes and edges, as well as topology, structure and hierarchy. Animated diagrams lead to cognitive overload when trying to detect trends, but are good at tracing particular nodes in a graph over time.[3] One way to use static displays is to use a small multiples approach where different versions are displayed next to each other.

## 5 Software Evolution Visualization

While our focus is on visualization for ontology evolution, a broader area of research is software visualization [14]. Software evolution visualization is a specific area of focus within software visualization and is defined as the process of visualizing the evolution of software by representing how different aspects have changed over time [14]. A comprehensive systematic mapping study of software evolution visualization [38] found that most papers focused on the following areas: change comprehension, contribution analysis, reverse engineering, identification of anomalies, and development communication. Change comprehension, contribution analysis and reverse engineering focused most on visualization. The most common data source used in the visualizations comes from source configuration management tools and the code itself. We briefly review some techniques and connect them to the ontology evolution functionality in Table 1.

The most identified software evolution visualization technique was *change comprehension* which aims to understand how software has been changed in a given period of time, identify evolutionary patterns, or identify stable parts of the software (similar to 1.1-6, 1.9, 1.12-13). Many tools have explored different aspects of change comprehension. SeeSoft [16] was a seminal tool that explored how lines of code have changed over different versions (1.6). This tool displays all lines of a system for all different versions as minimized. Each line is a specific color and each time the line changes per version it is color coded differently. In [53] changes are represented in a time line (1.2). Semantic zooming allows users to analyze changes on different levels (raw, statement, method, type) (1.6). For complex changes rectangles in the time line represent the relative number of edits with respect to the entire file (height) as well as the time spent on the edits (width) (1.5). Chronicler [52] displays pieces of code as abstract syntax trees (AST) and builds a history graph that represents changes (insert, delete, split, merge) on individual nodes in the AST. Nodes can represent structures on different levels and history paths in the history graph represent changes to specific individual nodes (1.6).

*Contribution analysis* shows the activities of developers by visualizing who worked on the software (1.7) based on source code repositories like Git, SVN, and CVS. One of the most common ways to visualize contributors' changes is with graphs such as the Gevol tool [10]. Some tools have extended the graph-based approach by creating novel visualizations that include animation. Code_swarm [42] visualizes the contributions made by developers in version control systems represented as swarms and shows a histogram timeline of additions and deletions of files. Gource [7] visualizes contributions made by developers and uses a force directed layout to display the source files that are in the system and then have avatars of people flying through the visualizations to show what files they made changes to. A more recent example is Developer Rivers [6] which uses a timeline-based visualization technique to show developer contributions.

[3] This was also found in [3].

A common technique for *reverse engineering* software is to visualize source code with Polymetric Views [32]. The suite of Polymetric Views uses software metrics to represent the size of classes such as number of lines of code, number of classes, number of packages, number of methods, number of dependencies, and inheritance hierarchies (1.1). The Evolution Matrix [31] is one technique part of the Polymetric Views suite which visualizes the size of classes represented as a matrix. SourceVis [1] used Polymetric Views to show how the structure of systems, packages, and classes have changed over different versions and displayed on a large multi-touch tabletop (1.3). Some empirical studies have identified that the Polymetric View techniques are an effective technique for software evolution visualization using a 3D city metaphor and large visualization wall [2, 51].

## 6 Discussion and Conclusion

Visualization for ontology evolution can help engineers better understand how an ontology has evolved. Despite the existence of ontology evolution tools and many software evolution visualization tools and techniques, there is a lack of research on *ontology evolution visualization*. Our work aims to fill this void by developing ontology evolution visualization tools to help ontology engineers. As a first step in realizing this goal we identified a set of functional requirements for ontology evolution.

In this paper we did not focus on the visualization of ontology versions, as this bears similarity to ontology visualization (for overviews we refer to [26, 33, 35]). One of the challenges in front of ontology visualization techniques is representing the richness of the ontologies in a comprehensive and scalable way. Ontology evolution tools additionally need to represent the differences between two versions of the ontology in a comprehensive way on different levels of granularity. To provide an overview and an initial exploration point in the presence of many entities some tools provide statistics for the changes and further allow drilling down to individual change. Multiple connected views can be utilized to account for the complexity of the ontologies and the need of provenance information. Although relevant to ontology development as well, the demand for provenance information is even higher in the case of ontology evolution to support decision making and auditing tasks.

The reviewed systems usually do not cover the complete ontology evolution process. Steps 2-4 are often not addressed although there are other systems that can be used for these steps [54]. Regarding visualization of information, spreadsheets are often used for ontology version and change statistics, change logs as well as provenance information about changes. Also line charts, pie charts and tag clouds are used for change statistics. Changes are often visualized on indented tree visualizations of ontology versions or in a unix-style diff. Highlighting and color coding are used to visualize the changes. Indented trees and graphs are most often used to visualize ontologies and the systems we reviewed represent the ontology versions as such.

While there are many techniques for software evolution visualization, these techniques are yet to be applied effectively for ontology evolution. In particular we see adapting software visualization techniques in the areas of change comprehension, contribution analysis and reverse engineering as possible techniques to explore visualization for ontology evolution. For instance, approaches for showing version graphs and for showing changes such as matrices and polymetric views could be used and the contribution analysis approaches are largely missing in ontology evolution systems.

In the future we will investigate also the field of schema versioning and evolution. Although ontology evolution is not the same as schema evolution [40], some of the visualization techniques in the latter (e.g., [37]) may be useful for ontology evolution. As a next step we will implement different visualization techniques for the identified functionality and conduct user studies to evaluate their applicability.

**Acknowledgments.** We acknowledge the EU FP7 project VALCRI (FP7-IP-608142), the National Graduate School in Computer Science (CUGS), and the Swedish e-Science Research Centre (SeRC) for financial support.

## References

1. C Anslow, S Marshall, J Noble, and R Biddle. SourceVis: Collaborative software visualization for co-located environments. In *International Working Conference on Software Visualization*, VISSOFT, pages 1–10. IEEE, 2013.
2. C Anslow, S Marshall, J Noble, E Tempero, and R Biddle. User evaluation of polymetric views using a large visualization wall. In *International Symposium on Software Visualization*, SOFTVIS, pages 25–34. ACM, 2010.
3. B Bach, E Pietriga, and J-D Fekete. Graphdiaries: Animated transitions andtemporal navigation for dynamic networks. *IEEE Transactions on Visualization and Computer Graphics*, 20:740–754, 2014.
4. F Beck, M Burch, S Diehl, and D Weiskopf. A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum*, 2016.
5. M Burch and S Lohmann. Visualizing the evolution of ontologies: a dynamic graph perspective. In *International Workshop on Visualizations and User Interfaces for Ontologies and Linked Data*, volume 1456 of *CEUR Workshop Proceedings*, pages 69–76, 2015.
6. M Burch, T Munz, F Beck, and D Weiskopf. Visualizing work processes in software engineering with developer rivers. In *International Working Conference on Software Visualization*, VISSOFT, pages 116–124. IEEE, 2015.
7. A. Caudwell. Gource: visualizing software version control history. In *Proceedings of the OOPSLA Companion*, Splash, pages 73–74. ACM, 2010.
8. W Ceusters. Applying evolutionary terminology auditing to the gene ontology. *Journal of Biomedical Informatics*, 42:518529, 2009.
9. V Christen, A Gross, and M Hartung. Region Evolution eXplorer a tool for discovering evolution trends in ontology regions. *Journal of Biomedical Semantics*, 6:Article 26, 2015.
10. C Collberg, S Kobourov, J Nagra, J Pitts, and K Wampler. A system for graph-based visualization of the evolution of software. In *Symposium on Software Visualization*, SoftVis, pages 77–86. ACM, 2003.
11. M Da Silveira, JC Dos Reis, and C Pruski. Management of dynamic biomedical ontologies: Current status and future challenges. In *IMIA Yearbook of Medical Informatics*, pages 125–133. 2015.
12. O Dameron, C Bettembourg, and N Le Meur. Measuring the evolution of ontology complexity: The gene ontology case study. *PLOS ONE*, 8(10):Article e75993, 2013.
13. P De Leenheer and T Mens. Ontology evolution. In M Hepp, P De Leenheer, A De Moor, and Y Sure, editors, *Ontology Management*, pages 131–176. 2008.

14. S Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2007.
15. Z Dragisic, P Lambrix, and E Blomqvist. Integrating ontology debugging and matching into the extreme design methodology. In *6th Workshop on Ontology and Semantic Web Patterns*, 2015.
16. S Eick, J Steffen, and E Sumner. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, November 1992.
17. S M Falconer and M A Storey. A Cognitive Support Framework for Ontology Mapping. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC/ASWC*, volume 4825 of *LNCS*, pages 114–127, 2007.
18. G Flouris, D Manakanatas, H Kondylakis, D Plexousakis, and G Antoniou. Ontology change: Classification and survey. *Knowledge Engineering Review*, 23(2):117–152, 2008.
19. A Gross, M Hartung, K Prufer, J Kelso, and E Rahm. Impact of ontology evolution on functional analyses. *Bioinformatics*, 28(20):2671–2677, 2012.
20. M Hartung, A Gross, and E Rahm. CODEX: Exploration of semantic changes between ontology versions. *Bioinformatics*, 26(6):895–896, 2012.
21. M Hartung, A Gross, and E Rahm. COnto-Diff: Generation of complex evolution mappings for life science ontologies. *Journal of Biomedical Informatics*, 46(1):15–32, 2013.
22. M Hartung, T Kirsten, A Gross, and E Rahm. OnEX: Exploring changes in life science ontologies. *BMC Bioinformatics*, 10:Article 250, 2009.
23. M Hartung, T Kirsten, and E Rahm. Analyzing the evolution of life science ontologies and mappings. In *Data Integration in the Life Sciences, DILS*, volume 5109 of *LNCS*, pages 11–27, 2008.
24. V Ivanova and P Lambrix. A unified approach for aligning taxonomies and debugging taxonomies and their alignments. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC*, volume 7882 of *LNCS*, pages 1–15, 2013.
25. V Ivanova, P Lambrix, and J Åberg. Requirements for and evaluation of user support for large-scale ontology alignment. In *The Semantic Web. Latest Advances and New Domains, 12th European Semantic Web Conference, ESWC*, volume 9088 of *LNCS*, pages 3–20, 2015.
26. A Katifori, C Halatsis, G Lepouras, C Vassilakis, and E G. Giannopoulou. Ontology visualization methods - a survey. *ACM Computing Surveys*, 39(4), 2007.
27. M Klein, D Fensel, A Kiryakov, and D Ognyanov. Ontology versioning and change detection on the web. In *13th International Conference on Knowledge Engineering and Knowledge Management*, volume 2473 of *LNCS*, pages 197–212, 2002.
28. P Lambrix. Towards a semantic web for bioinformatics using ontology-based annotation. In *14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises - WETI-ICE*, pages 3–7, 2005.
29. P Lambrix and R Kaliyaperumal. A session-based ontology alignment approach enabling user involvement. *Semantic Web Journal*, 2016.
30. P Lambrix, F Wei-Kleiner, and Z Dragisic. Completing the is-a structure in light-weight ontologies. *Journal of Biomedical Semantics*, 6:Article 12, 2015.
31. M Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *4th International Workshop on Principles of Software Evolution*, IWPSE, pages 37–42. ACM, 2001.
32. M Lanza and S Ducasse. Polymetric views - a lightweight visual approach to reverse engineering. *Transactions on Software Engineering*, 29(9):782–795, 2003.
33. M Lanzenberger, J Sampson, and M Rester. Ontology visualization: Tools and techniques for visual representation of semi-structured meta-data. *Journal of Universal Computer Science*, 16(7):1036–1054, 2010.
34. S Leonelli, A Diehl, K Christie, M Harris, and J Lomax. How the gene ontology evolves. *BMC Bioinformatics*, 12:Article 325, 2011.

35. S Lohmann, S Negru, F Haag, and T Ertl. Visualizing ontologies with VOWL. *Semantic Web Journal*, 7:399–419, 2016.
36. J Malone and R Stevens. Measuring the level of activity in community built bio-ontologies. *Journal of Biomedical Informatics*, 46:5–14, 2013.
37. L Meurice and A Cleve. DAHLIA: A visual analyzer of database schema evolution. In *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering, CSMR-WCRE*, pages 464–468, 2014.
38. R Lima Novais, A Torres, T Souto Mendes, M Mendonça, and N Zazworka. Software evolution visualization: A systematic mapping study. *Information and Software Technology*, 55(11):1860–1883, November 2013.
39. NF Noy, A Chugh, W Liu, and MA Musen. A framework for ontology evolution in collaborative environments. In *The Semantic Web, 5th International Semantic Web Conference, ISWC*, volume 4273 of *LNCS*, pages 544–558, 2006.
40. NF Noy and M Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6:428440, 2004.
41. NF Noy and MA Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *18th National Conference on Artificial Intelligence, AAAI*, page 744750, 2002.
42. M Ogawa and K-L Ma. Code_swarm: A design study in organic software visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1097–1104, November 2009.
43. R Palma, F Zablith, P Haase, and O Corcho. Ontology evolution. In MC Suarez-Figuero, A Gomez-Prez, E Motta, and A Gangemi, editors, *Ontology Engineering in a Networked World*, pages 235–255. 2012.
44. C Pesquita and F Couto. Predicting the extension of biomedical ontologies. *PLOS Computational Biology*, 8:e1002630, 2012.
45. M Poveda-Villalon, A Gomez-Perez, and MC Suarez-Figueroa. Oops! (ontology pitfall scanner!): An on-line tool for ontology evaluation. *International Journal on Semantic Web & Information Systems*, 10(2):7–34, 2014.
46. P Shvaiko and J Euzenat. Ontology Matching: State of the Art and Future Challenges. *Knowledge and Data Engineering*, 25(1):158–176, 2013.
47. L Stojanovic, A Maedche, B Motik, and N Stojanovic. User-driven ontology evolution management. In *13th International Conference on Knowledge Engineering and Knowledge Management*, volume 2473 of *LNCS*, pages 285–300, 2002.
48. O Vrousgrou, T Burdett, H Parkinson, and s Jupp. Biomedical ontology evolution in the EMBL-EBI ontology lookup service. In *Workshop proceedings of the EDBT/ICDT 2016 Joint Conference*, volume 1558 of *CEUR Workshop Proceedings*, 2016.
49. H Wache, T Vögele, U Visser, H Stuckenschmidt, G Schuster, H Neumann, and S Hübner. Ontology-based integration of information - a survey of existing approaches. In *IJCAI Workshop on Ontologies and Information Sharing*, pages 108–117, 2001.
50. S Walk, Ph Singer, L Espin Noboa, T Tudorache, MA Musen, and M Strohmaier. Understanding how users edit ontologies: Comparing hypotheses about four real-world projects. In *The Semantic Web - 14th International Semantic Web Conference, ISWC*, volume 9366 of *LNCS*, pages 551–568, 2015.
51. R Wettel, M Lanza, and R Robbes. Software systems as cities: A controlled experiment. In *International Conference on Software Engineering*, ICSE, pages 551–560. ACM, 2011.
52. M Wittenhagen, C Cherek, and J Borchers. Chronicler: Interactive exploration of source code history. In *CHI Conference on Human Factors in Computing Systems*, pages 3522–3532, 2016.
53. YS Yoon and BA Myers. Semantic zooming of code change history. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 95 – 99, 2015.

54. F Zablith, G Antoniou, M d'Aquin, G Flouris, H Kondylakis, E Motta, D Plexousakis, and M Sabou. Ontology evolution: a process-centric survey. *The knowledge engineering review*, 30:45–75, 2013.