Using Contexts in Ontology Structural Change Analysis

Vadim Ermolayev[1], Anton Copylov[2], Natalya Keberle[1],
Eyck Jentzsch[3], Wolf-Ekkehard Matzke[3]

[1] Zaporozhye National University, Zhukovskogo st. 66 69063 Zaporozhye Ukraine
vadim@ermolayev.com, nkeberle@gmail.com
[2] Metrotech IT, 46 Gryaznova st. Zaporozhye Ukraine
anton.copylov@gmail.com
[3] Cadence Design Systems GmbH, Mozartstr 2 D.85622 Feldkirchen Germany
jentzsch@cadence.com, wolf@cadence.com

Abstract. This demonstration paper presents the Ontology Difference Visualizer software tool. This tool helps making a certain step forward in improving the productivity and the accuracy of ontology engineering work. It provides a visualization of the structural difference between the two compared ontology schemas in the extended UML class diagram notation. Such visualization is built based on the comparison of the ontology code in OWL-DL using the PROMPTDiff algorithm. Having the difference between the ontologies visualized we allow checking at the model level how the intended modification of the source ontology has been implemented in the target ontology. The particular focus of this demonstration is the use of the contexts – the structural partitions of an ontology. Using contexts allows making the task of the change analysis more focused and less complex. For that the functionality is provided for concentrating on the semantic neighborhoods of the analyzed concepts with respect to different requirements or different owners within the team of ontology engineers.

Keywords: ontology, structural difference, ontology context, visualization.

1 Introduction

Ontology engineering is the discipline that, among the other important objectives, is concerned about the development of the methods and tools for building ontologies so as to fulfill the requirements of the target users about the common sense or a target subject domain. As the analogy to the other engineering domains suggests, industrial engineers will seek for and accept a methodology that is well defined and based on the use of standardized working patterns. However the development of ontologies is still more a craft work or a non-trivial mental exercise than a rigorous and standardized engineering process.

An industrially strong engineering methodology must contain a well defined and rigorous verification workflow – to check if the requirements have been correctly and fully implemented in the developed artifact. Such a flow will only be efficient if supported by a tool that releases an engineer from the burden of routine and effort consuming operations. For instance, if an iterative ontology development process is in operation, there is a need to present the differences between the source and the target version of the ontology. The difference has to be presented in a form that could be easily comprehended and analyzed by the verification engineer. His task then is to compare the generated difference report with the requirements to the target ontology version and to judge about the correctness and completeness of the implementation.

Verification becomes even more important when the artifact is developed collaboratively by several engineers or even distributed teams working on its different blocks. In this case the result needs to trace the errors to the proper context of the engineer who owns the development of the fragment in focus.

The objective of this paper is to demonstrate the tool for helping verifying ontology engineering work – Ontology Difference Visualizer (ODV). The proof of concept prototype of ODV has been developed to support the verification steps of the Shaker Modeling Methodology for ontology refinement [1], [2] in the PSI project[1]. The design choices for the prototype are explained based on the review of the related work in Section 2. ODV generates the report about the structural difference in the pair of ontologies and visualizes this difference using the extended notation of UML class diagrams as presented in Section 3. ODV allows analyzing the difference by appropriate parts that are called ontological contexts as presented in Section 4. The operational context model of ODV is based on the context model of PSI Upper-Level Ontology [1]. Section 5 presents the ODV demonstration that uses the pair of versions of the PSI Core Time ontology – v.2.2 and v.2.3. This pair of ontology versions has been used in one of our tool evaluation and validation experiments. Finally, the paper is concluded and the plans for the future work are outlined in Section 6.

2 Related Work

The central part of the presented work is the visualization of ontology changes. Therefore, the related work to be analyzed is the results in ontology visualization with the emphasis on the visualization of the structural changes in ontologies.

Methods and tools for ontology visualization (please refer to [3] for a detailed review) are classified by the basic representation style into the following categories: indented list (Protégé[2], OntoEdit [4]); node-link (OntoViz[3], OWLViz[4], OntoSphere [5]); zoomable view (Jambalaya [6], CropCircles [7]); space filling (treemap visualization of gene ontologies [8]); focus plus context or distortion (hyperbolic view in TGVizTab [9]). Currently the tools for ontology development more often use a 2D visualization metaphor. Examples are OWLViz, TGVizTab, Jambalaya plug-ins for Protégé. 3D visualization approach is also investigated in, for example, [10], OntoSphere [5]. OntoSphere is a Protege plug-in using 3D visualization metaphor and interaction in order to intelligently navigate through the dense information space. In

[1] Performance Simulation Initiative (PSI) is the internal R&D project of Cadence Design Systems GmbH.

[2] Protégé Project. Stanford University, http://protege.stanford.edu

[3] OntoViz plug-in for Protégé, http://protegewiki.stanford.edu/wiki/OntoViz

[4] OWLViz plug-in for Protégé, http://protegewiki.stanford.edu/wiki/OWLViz

the prototype presented in [10] the sphere-packing technique is used instead of 2D circle-filling (used in CropCircles [7]) in order to combine the 3D navigation (rotation, semantic zoom) with the fine tuning of the level of details.

Standardization efforts in visualizing ontologies are mainly within OMG [11]. OMG elaborates the UML meta-model for the representation of ontologies encoded among others in RDF and OWL. Since definition of UML Ontology Definition Metamodel for OWL DL in [12], UML-based visualization of ontologies is used both in commercial tools like TopBraid[5] Composer™, Sandpiper[6] Software Visual Ontology Modeler™ and in freely available tools, e.g. in OWLGrEd [13].

Our ODV presented in this paper uses UML-based notation for representing the visualizations of structural changes in ontologies. The basic notation has been chosen because of the standardization efforts of OMG and the use of UML-based visualizations of ontologies in some commercial tools. One more argument in favor of the UML-based notation was that it is widely accepted across disciplines as a visual language for modeling and design. Therefore, its use helps decreasing the ramp-up efforts of both the ontology engineers and the subject experts to start understanding this visual ontology representation. We have proposed the extension of the UML class diagram notation that is presented in Section 3.

Unfortunately there are not many approaches and tools for visualizing structural differences in ontologies. One relevant approach implemented in the software tool is the PROMPT-Viz[7] Protégé plug-in [14], relying on PROMPTdiff [15] in calculating structural difference between ontologies. The visualization combines hierarchically embedded rectangles – nested treemaps – with a zoomable user interface. The treemap notation can only be effectively used for hierarchical structures in ontologies, and can not be easily extended. Another example is the visualization of differences in indented lists in OWLDiff[8].

With respect to computing structural differences in ontologies several research results, comprising algorithms and software prototypes, need to be mentioned. However we omit a detailed discussion of this aspect here as the focus of this paper is visualization and refer to our technical report [16] for a detailed comparative analysis of these algorithms and software tools. The ODV uses the PromptDiff algorithm [15] for computing structural difference.

3 Extended UML Notation for Structural Change Visualization

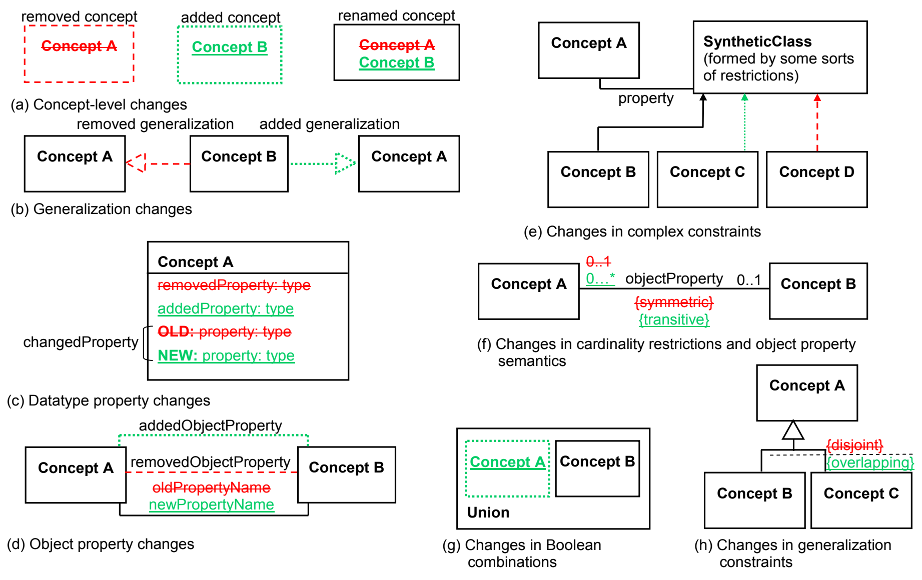
In the developed extension of the UML class diagram notation for structural change visualization, every change is presented as a set of atomic sub-changes of two types – added or removed elements. The added elements are marked green and the removed – red. In the cases of compound changes the diagram contains the minimal information about such a change and also comprises every element required for the visualization of the change.

[5] TopQuadrant TopBraid Composer, http://www.topquadrant.com/products/TB_Composer.html
[6] Sandpiper Software Visual Ontology Modeler, http://www.sandsoft.com/products.html
[7] PROMPT-Viz plug-in for Protégé, http://protegewiki.stanford.edu/wiki/PromptViz
[8] OWLDiff plug-in for Protégé, http://protegewiki.stanford.edu/wiki/OWLDiff

OWL classes are represented by UML classes. The removed OWL classes are rendered with red border and class name. The added OWL classes – with green border and class name. If a class is renamed both the old name (red) and the new name (green) appear in the class box (Fig. 1a).

OWL generalization changes are visualized as colored UML generalizations (Fig. 1b). Added or removed generalizations to OWL: Thing class are not displayed.

OWL datatype properties are presented as UML class attributes. The range of a property forms the type of the corresponding attribute, the property name determines the name of the corresponding attribute and the domain determines the class-holder of the attribute. Added datatype properties are marked green, removed datatype properties are marked red. Renamed attributes are visualized as the combination of the added attribute with the new name and the removed attribute with the old name to provide the complete information about the change (Fig. 1c). So far the notation does not allow rendering all the possible changes in datatype properties – for example the change of the type of an attribute (OWL property range). The refinement is planned for the future work.



Legend: The following denotation for colors has been introduced to provide the readability of the diagrams in the black and white mode.
Text labels: ~~**strikedthrough**~~ labels correspond to red; **underlined** labels correspond to green.
Lines and borders: dashed lines (- - - - -) correspond to red; dotted lines (.............) correspond to green

Fig. 1. The notation for ontology structural change visualization (extension of UML).

Object properties are presented as UML associations. When a property doesn't have the inverse property its name becomes the association name. When a property has the inverse property, their names are used as association roles. Deleted object properties are visualized as associations marked red while added ones are marked green. Renamed properties are visualized as associations with two names – the old name marked red and the new name marked green (Fig. 1d).

Domain and range restriction changes are rendered as added or removed associations. Unnamed classes formed by complex restriction statements are represented by synthetic classes. In such cases only the changed parts of a synthetic class are marked with color. Nevertheless, the whole synthetic class must be shown to ensure the completeness of the change diagram (Fig. 1e).

Changes to cardinality restrictions are represented as cardinalities of associations marked with corresponding color (Fig. 1f). The changes in functional, inverse functional, symmetric, and transitive statements for object properties are represented as the semantics of associations, marked green for additions and red for deletions (Fig. 1f).

Enumeration, union, or intersection axioms are rendered as quads containing the set of internal UML elements. Each quad consists of the name of the logical operation and the set of operands (classes or individuals). Changed elements in the quad are marked green (added) or red (removed). The unchanged elements are also shown (Fig. 1g).

Generalization constraints are displayed as per the UML standard. The removed constraints are rendered red and the added ones are green (Fig. 1h).

The presented notation for ontology change has been deliberately restricted to render the structural changes in ontology schemas (TBox). The reason is that the software tool for analyzing differences (ODV) has been developed to support the particular phase of Ontology Refinement and Validation in the Shaker Modeling methodology for ontology refinement. Structural changes where the instances of ontologies are involved are analyzed within a different methodology phase – ontology ABox Migration (Fig. 4). This analysis is supported by a different software tool we developed [23].

4 Context Model and ODV Functionality

ODV allows analyzing the difference between the compared ontologies by appropriate parts that are called ontological or structural contexts. The formal model for representing contexts is the one of the PSI Upper-Level Ontology [17]. Here the context model and its implementation and use in the ODV are briefly presented.

As only the structural part of the context model is relevant for ODV (Fig. 2, dark grey concepts), the process related part of the upper-level model is not in use. The relevant part of the formal model could be interpreted as follows.

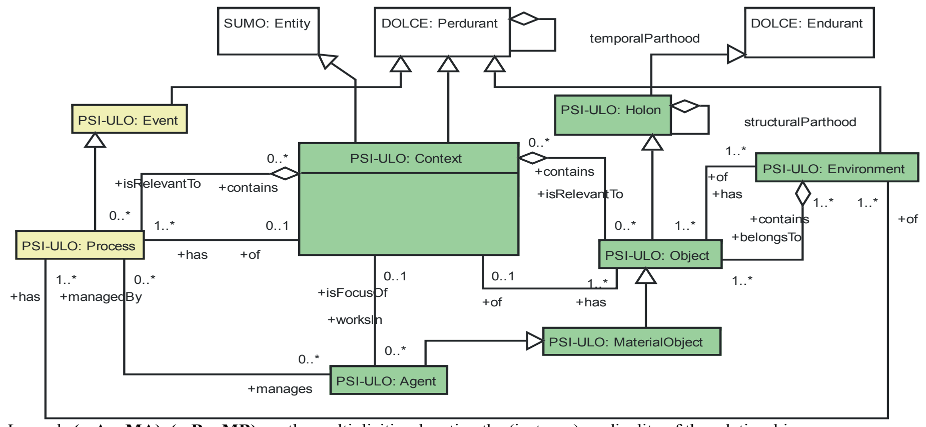
A Context[9] of an entity (that has the context) is the selection of related things that facilitate interpreting, using, or performing the entity having this context in a pragmatic way. For denoting a Context it is therefore required to answer:

(i) **The context of what is specified?** A Context could be either of a Process or of an Object. For ODV the context of a Process is not relevant while the Objects for ODV are the concepts of the analyzed ontologies. So, we are interested in the contexts of ontology concepts.

(ii) **What is relevant for the inclusion in the context?** An instance of a Context may aggregate the instances of a Process or of an Object as relevant referential

[9] http://isrg.kit.znu.edu.ua/ontodocwiki/index.php/Context

constituents. For ODV only Objects are relevant constituents where the instances of an Object would be the concepts of the analyzed ontologies together with all their properties.



Legend: **(mA…MA), (mB…MB)** are the multiplicities denoting the (instance) cardinality of the relationship:
(m,M)A specifies how many (**m**inimally, **M**aximally) instances of **A** may be related to the ONE SPECIFIC instance of **B**;
(m,M)B specifies how many (**m**inimally, **M**aximally) instances of **B** may be related to the ONE SPECIFIC instance of **A**.
DOLCE is the foundational ontology of the WonderWeb ontology library [18]; SUMO is the Suggested Upper Merged Ontology [19]; PSI-ULO is the PSI Upper-Level Ontology [1]

Fig. 2: The upper-level context model [17] used in ODV.

(iii) **Who uses the Context?** A Context is used by an Agent to define the current working focus and determine working priorities. For ODV the instance of an Agent will be the user (the ontology engineer) who is the owner of the concept whose structural context is described.

Fig. 3 presents the desktop of the ODV. The composition of a context of a concept (0), as implemented in the ODV, could be formed by specifying the radius of the neighborhood of this concept (1). Further it could be fine-tuned by manual inclusion or exclusion of the concepts (2), object properties (3), subsumption relationships (4). The analyzed ontological context may be placed on the wafer of the source (old) ontology by toggling the “show old” mode in the Tools menu. The context may be also altered by considering or filtering out the concepts belonging to the imported ontology modules (5). Finally the “owner” filter may be employed for concentrating on the changes that have been introduced by a particular ontology engineer in the team (6).

The ODV implementation allows also editing and saving the layout of the visualized structural difference in the project file (7). Such a layout saves all the context settings and therefore allows personalized representations for different users.

The reported release of the ODV proof of concept software prototype has been implemented in Java as a plug-in to the Cadence ProjectNavigator software prototype. ODV uses OWL API 2 and therefore is capable of processing OWL ontologies coded in OWL DL. OWL DL has been used in the ontologies that have been analyzed in ODV evaluation experiments.

5 ODV Demonstration

For the demonstration of the ODV the pair of the PSI Core Time ontology versions (2.2 and 2.3) has been selected. The reasons for the selection of this particular pair of ontologies are: (i) the ontologies are small enough for being effectively demonstrated; (ii) structural changes were substantial and the types of changes in this pair of ontology versions cover almost all the spectrum of the change notation; and (iii) the ontology engineering work on the refinement of the Time ontology has been documented in line with the phases of the ontology engineering methodology [2] – this documentation may be used as a back-up demonstration material.

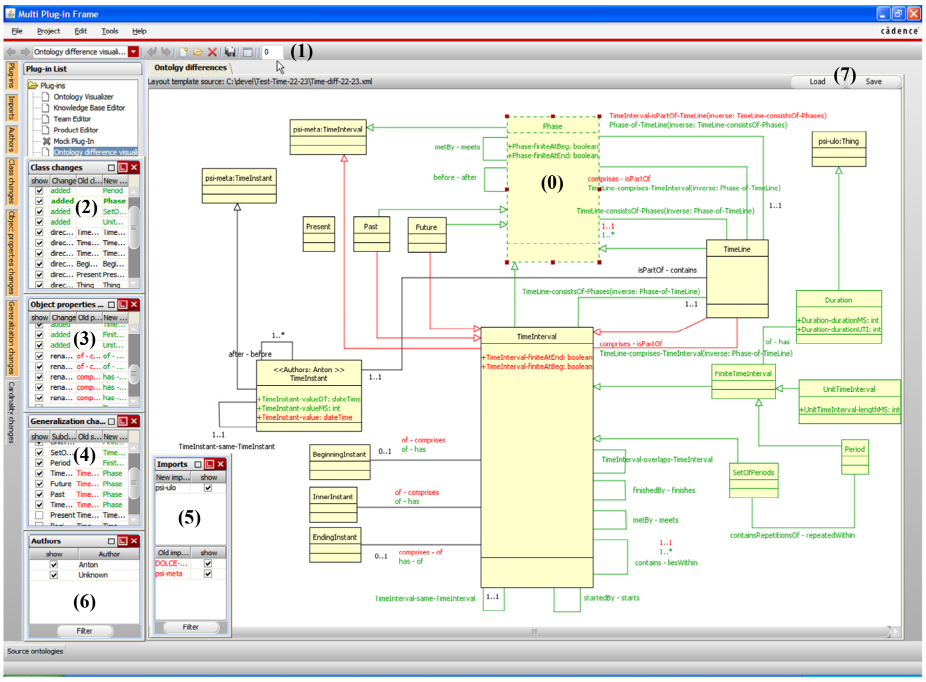


Fig. 3: The ODV desktop. Visualized is the structural difference between the PSI Time Core ontologies v.2.2 and v.2.3.

The primary reason for the change requirements to the Time ontology v.2.2 was that the underlying minimal model of time [20] was not capable of meeting the requirements for the development of the design project scheduling component for the Cadence ProjectNavigator [21]. Therefore it has been decided that the formal model of time is upgraded to the time crisp model [20] and the ontology will be refined so as to provide the descriptive models for: (i) the phases of time intervals; (ii) periods – the finite time intervals associated to the particular repetitions of events; and (iii) the sets of periods – the collections of periods describing the part of the time line corresponding to all the repetitions of the event. These requirements have been implemented in the Time ontology v.2.3[10].

Following the Shaker Modeling Methodology, the team comprising two ontology engineers and two subject experts: have specified the requirements; have refined the formal model of time; have proposed the UML model of the proposed solution; have discussed this model, refined it and agreed on the release candidate; have implemented the ontology in OWL-DL based on the agreed model. After performing these development steps the changes in the ontology have been verified using the ODV (Fig. 4). The first iteration of the change analysis revealed that though all the requirements have been accounted for in the proposed UML model, the OWL-DL ontology code did not implement all the required changes. These errors have been easily detected using context filtering. The knowledge engineer who was the owner of the context containing the error has been tasked to provide the correction. The next iteration of the change analyses revealed no errors.

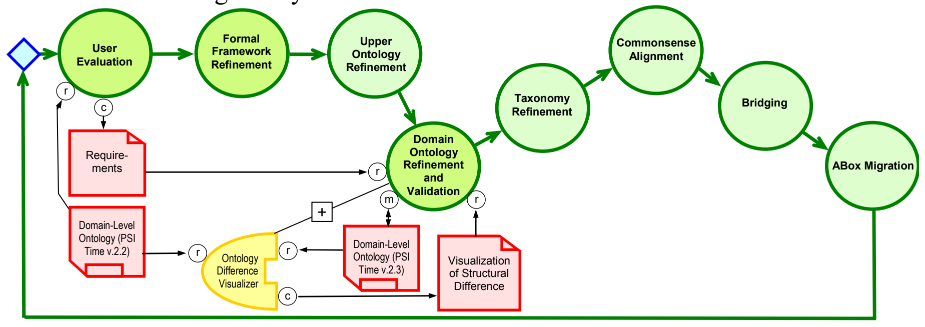

User Evaluation
Formal Framework Refinement
Upper Ontology Refinement
Domain Ontology Refinement and Validation
Taxonomy Refinement
Commonsense Alignment
Bridging
ABox Migration
Require-ments
Domain-Level Ontology (PSI Time v.2.2)
Ontology Difference Visualizer
Domain-Level Ontology (PSI Time v.2.3)
Visualization of Structural Difference
r
c
m
+

Fig. 4. The use of ODV to support the Shaker Modeling Methodology for ontology refinement. The methodology is presented using the ISO/IEC 24744 notation [22].

As it could be seen in Fig. 4, the ODV is the tool that supports the verification steps of the ontology refinement phases of the methodology. It computes and visualizes the structural difference between the older version of the ontology that has been developed before (in the previous methodology iteration) and the newer version of the ontology. The visualization diagram can be stored as an image and further used as a report for documenting the process of ontology engineering. The verification engineer manually compares the requirements produced in the user evaluation phase to the generated difference diagram to check if all the requirements have been properly implemented. We have observed in the evaluation experiments that the contextualization features of the ODV noticeably increase the performance of the validation engineer in executing this important operation.

6 Concluding Remarks and Outlook

The paper supports the demonstration of the ODV software tool. Emphasis has been put on the explanation of how this tool uses ontological contexts for making the

[10] http://isrg.kit.znu.edu.ua/ontodocwiki/index.php/PSI_Core_Time_ontology

collaborative work of ontology engineers more productive – in particular in the verification step of ontology refinement. The ODV computes the structural difference between the pair of ontologies and renders this difference using the presented extension of the UML class diagram notation. The layout and the rendered ontological context may be altered and saved in a project file thus enabling personalized views on the parts of the ontologies to be verified.

For computing structural differences the ODV uses the PromptDiff fixpoint algorithm mainly because of its modularity and extendibility. For the future it is planned that several heuristic matches will be developed as extensions. At the moment the ODV is a single user prototype that uses local file system as the ontology repository. As the tool is thought as an instrument for collaborative work in a distributed team it will be further developed to enable multiuser work with a centralized triple store. Furthermore, the ODV may reveal its full potential as a verification and validation instrument only as a component of an ontology model (schema) editor. The development of such an editor and the integration of the upcoming revisions of ODV into it are also the plans for our future work.

References

1. Ermolayev, V., Keberle, N., Matzke, W.-E.: An Upper-Level Ontological Model for Engineering Design Performance Domain. In: Li, Q., Spaccapietra, S., Yu, E. (eds.) ER 2008. LNCS, vol. 5231, pp. 98--113. Springer, Heidelberg (2008)
2. Ermolayev, V., Jentzsch, E., Keberle, N., Sohnius, R.: Performance Simulation Initiative. Shaker Modeling Methodology for Ontology Refinement. Reference Specification v.1.0. Technical Report PSI-METH-TR-2009-1, 75 p. VCAD EMEA Cadence Design Systems, GmbH (2009)
3. Katifori, A., Halatsis, K., Lepouras, G., Vassilakis, C., Giannopoulou, E.: Ontology Visualization Methods – A Survey. ACM Computing Surveys, 39(4) (2007)
4. Sure, Y., Angele, J., Staab, S.: OntoEdit: Guiding ontology development by methodology and inferencing. In: Meersman, R., Tari, Z. (eds.) DOA/CoopIS/ODBASE 2002, LNCS, vol. 2519, pp. 1205--1222. Springer, London (2002)
5. Bosca, A., Bonino, D., Pellegrino, P.: OntoSphere: more than a 3D ontology visualization tool. In: 2nd Italian Semantic Web Workshop (2005)
6. Storey, M.-A., Noy, N., Musen, M. A., Best, C., Fergerson, R., Ernst, N.: Jambalaya: an interactive environment for exploring ontologies. In: 2002 International Conference on Intelligent User Interfaces, p. 239. ACM Press, New York, NY, USA (2002)
7. Wang, T. D., Parsia, B.: CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies. In: Cruz, I. F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC-2006. LNCS, vol. 4273, pp. 695--708. Springer, Heidelberg (2006)
8. Baehrecke, E. H., Dang, N., Babaria, K., Shneiderman, B.: Visualization and Analysis of Microarray and Gene Ontology Data with Treemaps. BMC Bioinformatics, 5, pp. 5--84 (2004)
9. Alani, H.: TGVizTab: An Ontology Visualisation Extension for Protégé. In: Workshop on Visualization Information in Knowledge Engineering at International Conference on Knowledge Capture (K-CAP-2003), http://eprints.ecs.soton.ac.uk/8326/
10. Dmitrieva, J., Verbeek, F. J.: Node-Link and Containment Methods in Ontology Visualization. In: Hoekstra, R., Patel-Schneider, P. F. (eds.) 6th International Workshop

"OWL: Experience and Directions" (OWLED-2009), http://ceur-ws.org/Vol-529/owled2009_submission_9.pdf

11. Ontology Definition Metamodel (ODM), Version 1.0 Specification, Object Management Group, Inc., Needham, MA, May 2009. Available at: http://www.omg.org/spec/ODM/1.0/
12. Brockmans, S., Volz, R., Eberhart, A., Löffler, P.: Visual Modeling of OWL DL Ontologies Using UML. In: Gil, Y., Motta, E., Benjamins, V. R., Musen, M. A. (eds.) ISWC 2004, LNCS, vol. 3298, pp. 198--213. Springer, Heidelberg (2004)
13. Bārzdiņš, J., Bārzdiņš, G., Čerāns, K., Liepiņš, R., Sproģis, A.: OWLGrEd: a UML Style Graphical Notation and Editor for OWL 2. In: Clark, K., Sirin, E. (eds.) Proc. 7th International Workshop "OWL: Experience and Directions" (OWLED-2010), http://www.webont.org/owled/2010/papers/owled2010_submission_5.pdf
14. Perrin, D. S. J.: PROMPT-Viz: Ontology Version Comparison Visualizations with Treemaps. Master Of Science Thesis, Department of Computer Science, http://hdl.handle.net/1828/528. University of Victoria, British Columbia, Canada (2004)
15. Noy, N., Musen, M.: PROMPTdiff: A Fixed-Point Algorithm for Comparing Ontology Versions. In: National Conference on Artificial Intelligence (AAAI/IAAI'02), pp. 744--750. AAAI Press (2002)
16. Copylov, A., Ermolayev, V., Keberle, N.: Performance Simulation Initiative. Ontology Revision Visual Analysis. Technical Report PSI-ONTO-RM-TR-2009-01, 34 p. VCAD EMEA Cadence Design Systems, GmbH (2009)
17. Ermolayev, V., Ruiz, C., Tilly, M., Jentzsch, E., Gomez-Perez, J. M.: A Context Model for Knowledge Workers. In: Ermolayev, V., Gomez-Perez, J.M., Haase, P., Warren, P. (eds.) 2nd International Workshop on Context, Information and Ontologies (CIAO 2010), http://ceur-ws.org/
18. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: WonderWeb Deliverable D18 Ontology Library (final), ICT Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web (2003)
19. Niles, I., Pease, A.: Towards a Standard Upper Ontology. In: International Conference on Formal Ontologies in Information Systems, vol. 2001, pp. 2--9. ACM Press, New York, NY, USA (2001)
20. Ermolayev, V., Keberle, N., Matzke, W.-E., Sohnius, R.: Fuzzy Time Intervals for Simulating Actions. In: Kaschek, R., Kop, C., Steinberger, C. and Fliedl, G. (eds.) IUNISCON 2008. LNBIP, vol. 5, pp. 429--444. Springer, Heidelberg (2008)
21. Ermolayev, V., Dengler, F., Jentzsch, E., Matzke, W.-E.: Articulation and Sharing of Distributed Design Project and Process Knowledge. In: In: M. Essaaidi et al. (eds.): Intelligent Distributed Computing IV, SCI 315, pp. 209--216, Springer Berlin/Heidelberg (2010)
22. Henderson-Sellers, B., Gonzalez-Perez, C.: Standardizing Methodology Metamodelling and Notation: An ISO Exemplar. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 1--12. Springer, Berlin/Heidelberg (2008)
23. Davidovsky, M., Ermolayev, V., Matzke, W.-E., Tolok, V.: Evaluation of Semi-Automated Ontology Instance Migration. In: M. Essaaidi et al. (eds.): Intelligent Distributed Computing IV, SCI 315, pp. 179--190, Springer Berlin/Heidelberg (2010)