Efficient Management of Biomedical Ontology Versions

Toralf Kirsten[1,2], Michael Hartung[1], Anika Groß[1], and Erhard Rahm[1,3]

[1] Interdisciplinary Centre for Bioinformatics, University of Leipzig
[2] Institute for Medical Informatics, Statistics and Epidemiology, University of Leipzig
[3] Department of Computer Science, University of Leipzig
{gross,hartung,tkirsten}@izbi.uni-leipzig.de,
rahm@informatik.uni-leipzig.de

Abstract. Ontologies have become very popular in life sciences and other domains. They mostly undergo continuous changes and new ontology versions are frequently released. However, current analysis studies do not consider the ontology changes reflected in different versions but typically limit themselves to a specific ontology version which may quickly become obsolete. To allow applications easy access to different ontology versions we propose a central and uniform management of the versions of different biomedical ontologies. The proposed database approach takes concept and structural changes of succeeding ontology versions into account thereby supporting different kinds of change analysis. Furthermore, it is very space-efficient by avoiding redundant storage of ontology components which remain unchanged in different versions. We evaluate the storage requirements and query performance of the proposed approach for the Gene Ontology.

Keywords: Ontology versioning, integration, management.

1 Introduction

Many ontologies have recently been developed and are frequently used in life sciences and other domains. In particular, ontologies are used to annotate resources by semantically describing their properties. For instance, molecular-biological objects are annotated with concepts of the well-known Gene Ontology (GO) [4] to describe their function or to specify the biological processes the objects are involved in. Moreover, personal properties of patients such as diseases can be described by concepts of medical ontologies, e.g., OMIM (http://www.ncbi.nlm.nih.gov/omim) or NCI Thesaurus [12]. Many analysis studies utilize such ontology-based annotations to better understand the real-world impact of certain observations. For instance, GO annotations are used for functional profiling [1] of large gene expression datasets to determine the semantic function of certain sets of heavily expressed genes.

Most ontologies especially in life sciences are frequently changed to capture new insights or correct previous specifications [5, 14]. Such evolutionary changes typically include the addition, deletion and modification of concepts, relationships, and attribute values/descriptions. These changes are incorporated in newer ontology versions that

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 574–583, 2009.

are released periodically. Current analysis studies do not consider the ontology changes reflected in different versions but typically limit themselves to the most current ontology version which may, however, become obsolete to a larger degree within a short period of time. Providing an easy access to different versions of an ontologies is currently not supported but would be helpful for applications to check the stability of analysis results and decide about the need to re-run earlier analysis studies.

There has been comparatively little previous research on efficient ontology versioning for large ontologies [3]. Klein and Fensel [8] proposed requirements for ontology versioning, in particular identification, change tracking and transparent evolution. As one task of ontology versioning some ontology management systems support the detection of differences between ontology versions (DIFF operation). The PROMPTDIFF approach [9] uses heuristics which automatically determine structural differences for OWL-based ontologies. OntoView [7] and OBO-Edit [2] include similar algorithms that produce information at the level of elementary changes, e.g., added and deleted ontology concepts. SemVersion [13] provides structural and semantic versioning of RDF-based ontologies with support for different triple stores. However, these systems do not explicitly utilize the information about changes for an efficient storage of ontology versions especially in case of large ontologies.

Compared to previous work our method focuses on the efficient storage and management of many versions of large biomedical ontologies. We utilize the observation that most succeeding versions differ only to a smaller extent to achieve an efficient ontology storage with a minimum of redundancy between different versions. We make the following contributions in the paper:

- We propose a scalable and space-efficient approach to integrate and manage many versions of different ontologies in a central repository providing applications a uniform ontology access. We have successfully applied our approach to large biomedical ontologies but it is generic and thus can also be applied for ontologies of other domains.
- Our approach includes an ontology versioning model that is the basis for an efficient ontology management. It also allows the identification of changes for concepts, relationships and attributes across different ontology versions. Compared to seeing static ontology versions the versioning model provides a dynamic view in which ontology elements are managed (versioned) according to their life time.
- We evaluate the storage efficiency and query performance of our approach on ontology version management utilizing various versions of the popular Gene Ontology.

The presented approach is fully operational and has been used in diverse applications. Firstly, the Ontology Evolution Explorer (OnEX, online access under http://www.izbi.de/onex) [6] is built upon the versioning system and provides different kinds of ontology change exploration and analyses. It currently integrates approx. 560 versions of 16 life science ontologies. Secondly, the ontology version system is used to study the impact of ontology evolution on functional profiling results using the FUNC package [11].

The rest of the paper is organized as follows. In Section 2 we introduce our versioning model and corresponding relational repository schema. Section 3 describes a method for assessing changes when a new ontology version will be imported. Results

of our performance analysis are illustrated in Section 4. We summarize and conclude in Section 5.

2 Versioning Model

We assume that an ontology $O=(C,R)$ consists of a set of concepts $C=\{c_1,\ldots,c_n\}$ and a set of relationships $R=\{r_1,\ldots,r_m\}$. Concepts represent the entities of a domain to be modeled and are interconnected with relationships of R. Each relationship is associated with a specific type, such as '*is-a*' or '*part-of*', describing the semantics of the relationship. Concepts and relationships form together the structure of the ontology which can range from a flat vocabulary over a hierarchical representation to a directed acyclic graph (DAG). Each concept is described by a set of attributes, such as concept ID, name, definition or description. We assume that each concept is uniquely identified by the concept ID; in life sciences this identifying attribute is mostly called *accession number*.

An ontology is usually released by the ontology provider at specific points in time to provide the current state of the ontology as a designated version. An ontology version $O_v=(C_v,R_v,t)$ of the ontology O in version v at timestamp t is defined by a set of concepts $C_v \subseteq C$ and a set of relationships $R_v \subseteq R$ that are valid at the creation time t of O_v. Ontology versions follow a linear version schema, i.e., there is a chain of ontology versions, such that each version v_i has at most one predecessor v_{i-1} and one successor v_{i+1}. To find out the validity of concepts at a specific point in time, we associate with a *life time* (t_{start}, t_{end}) with each concept and each relationship. A life time period begins with the date t_{start} of the ontology version in which the concept or relationship occurred for the first time. It ends with a date t_{end} of the ontology version in which the concept or relationship has been valid for the last time ($t_{start} < t_{end}$). Therefore, the set of concepts C_v (set of relationships R_v) of an ontology version O_v released at time t consists of all concepts $c_i \in C$ (relationship $r \in R$) which satisfy $t_{start} \leq t \leq t_{end}$.

In addition to the versioning of concepts and relationships at the ontology level, our versioning model also supports versioning for attributes at the concept level. At this level, we consider two aspects, a) the non-empty attribute set that is used to describe an ontology concept and b) the associated values per attribute. Associating the specific values of a concept attribute with a life time, i.e., a start and end date, allows selecting the concrete attribute value that is valid at a specific point in time t.

The version model has been realized within a generic database repository to store the versions of different ontologies and supporting uniform version access. The relational repository schema for the versioning model is shown in Fig. 1. The schema consists of different entities to consistently represent ontologies (entity *Ontologies*), ontology versions (entity *Ontology Versions*), concepts (entity *Ontology Concepts*) and the ontology structure (entity *Ontology Relationships*). The latter entity represents the set of relationships; each of them is associated with a relationship type (e.g., is-a, part-of). To flexibly store attribute values for each ontology concept we apply the entity-attribute-value concept (EAV) [10]. In this way, the entity *Attributes* holds the set of attributes describing the semantics of the associated attribute values (entity *Element Attribute Values*) of an ontology concept. Hence, new attributes and attribute values can be inserted without changing the repository schema. Each ontology concept, relationship, and attribute

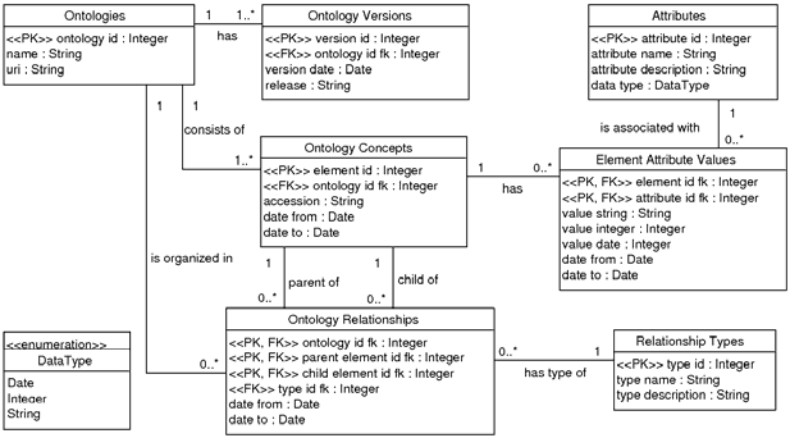

Fig. 1. Relational repository schema to efficiently manage ontology versions

value is associated with a life time represented by the fields '*date-from*' and '*date-to*'. Using these time periods, any ontology version can be completely reconstructed by taking the '*version date*' attribute (entity *Ontology Versions*) into account, such that it holds *date-from* $\leq$ *version date* $\leq$ *date-to*. Actually, one can determine the snapshot of an ontology for any point in time after the creation date.

3 Data Import and Change Detection

Given the frequent release of new versions it is important that new versions be automatically imported into the central version management system (repository). Furthermore, ontology changes compared to the previous version should automatically be detected to adapt the lifetime information of ontology components. The data flow for importing new versions from different ontology sources consists of three subprocesses, namely *download and import* of raw ontology versions, *change detection*, and *loading changed objects* into the repository.

Most ontology versions can automatically be integrated depending on the physical source type of an ontology. While some ontology providers store versions within a concurrent version system (e.g., CVS and SVN), other ontology versions can be downloaded from web pages or publicly accessible FTP directories. *Download-wrappers* for different physical source types regularly look up and automatically download data whenever a new version of a specified ontology is available. A new ontology version is then temporarily imported into a so-called staging area for further processing. *Format-specific importers* are used to import ontology versions taking the ontology representation into account. While OWL is the dominant ontology representation in the Semantic Web, many biomedical ontologies are provided in the OBO format or other formats, such as CSV, XML, and relational databases. We combine several format-specific importers with the download-wrappers to avoid ontology-specific wrapper and import implementations. During the download and import phase the version metadata including the version number and the creation timestamp of an

ontology version is recognized. This metadata is often encapsulated in file names especially when ontology versions are made available on web pages. In other cases, this metadata are available in specific documentation files or can be retrieved from systems such as CVS.

The version metadata is the basis for the *change detection phase* which derives the dynamic life time information from the static ontology versions. This transformation includes the following steps. Firstly, the predecessor Ov_{i-1} is identified for the given import ontology version Ov_i. Secondly, this version is compared with the import version to determine concepts and relationships which have been *added* and those that were *deleted*. This comparison is primarily performed by using the accession numbers (ID attribute) of ontology concepts. Finally, the detected changes are used to update the existing repository content. Newly introduced objects are added to the repository with a life time start equal to the timestamp of the import version. For deleted concepts and relationships the life time end is set to the date where an object was available for the last time which is normally the day before the import version O_{vi} was released.

The life time of added and deleted concept attributes is adapted similarly than for the addition and deletion of concepts/relationships. Changes on attribute values are treated in the following way. The old attribute value stored in the repository is annotated with an end timestamp to indicate the end of its validity period. Furthermore, a new attribute entry is created with the modified value and a life time start equal to the import version.

The example in Fig. 2 illustrates repository updates for selected changes between the May and June 2007 versions of the GO Cellular Components ontology. The concept GO:0009572 has been deleted, thus the end timestamps of this concept as well as its attributes and relationships are set to 2007-05. By contrast, the concept GO:0000446 with attributes (name, obsolete status, definition) as well as relationships to GO:0000347 and GO:0008023 were added. As a result the system creates new entries with a start timestamp of 2007-06. Finally, the name of GO:0009356 has changed from ‘p-aminobenzoate synthetase complex’ to ‘aminodeoxychorismate synthase complex’, hence the end timestamp of the corresponding attribute entry is set to 2007-05 and a new entry with start timestamp 2007-06 is inserted.

Concepts

accession number	*start*	*end*
GO:0009572	2002-02	**2007-05**
GO:0000446	**2007-06**	
...		

Relationships

source	*target*	*type*	*start*	*end*
GO:0009572	GO:0044459	is_a	2006-05	**2007-05**
GO:0009572	GO:0009510	part_of	2003-05	**2007-05**
GO:0000446	**GO:0000347**	**is_a**	**2007-06**	
GO:0000446	**GO:0008023**	**is_a**	**2007-06**	

Attributes

concept	*attribute*	*value*	*start*	*end*
GO:0009572	name	desmotubule central rod	2002-02	**2007-05**
GO:0009572	obsolete	false	2002-02	**2007-05**
GO:0000446	**name**	**nucleoplasmatic THO complex**	**2007-06**	
GO:0000446	**obsolete**	**false**	**2007-06**	
GO:0000446	**definition**	**The THO complex when ...**	**2007-06**	
GO:0009356	name	p-aminobenzoate synthetase complex	2002-12	**2007-05**
GO:0009356	**name**	**aminodeoxychorismate synthase complex**	**2007-06**	

Fig. 2. Extract of the versioning repository for GO cellular components

4 Evaluation

In this section we evaluate the storage efficiency and query performance of the presented approach for managing ontology versions. We first introduce different measures

for the storage and query evaluation. We then describe the evaluation scenario in the life science domain and analyze the evaluation results.

4.1 Evaluation Measures

To assess the storage requirements we determine the number of stored ontology elements. We also compare the storage requirements of our proposed versioning repository (*approach*) with the ones for a fully redundant storage of ontology versions (*naïve* version management). We use these measures for the number of elements in a particular ontology version:

$\|C\|_{vi}$, $\|R\|_{vi}$, $\|A\|_{vi}$	Number of concepts, relationships and attributes available in ontology version O_{v_i}
$\|E\|_{vi} = \|C\|_{vi} + \|R\|_{vi} + \|A\|_{vi}$	Number of elements available in ontology version O_{v_i}

We denote the first version by v_1 and the latest considered version by v_n. The overall number of elements stored in a repository when integrating n succeeding ontology versions O_{v1}, …, O_{vn} of an ontology can be calculated as follows.

$$\left|E\right|_n^{naive} = \sum_{i=1}^{n} \left|E\right|_{vi} \qquad \left|E\right|_n^{approach} = \left|\bigcup_{i=1}^{n} E_{vi}\right| \tag{1}$$

Since the naive approach completely stores each version, we need to sum up the elements available in each version to determine the overall number of elements for *n* versions. In contrast, our approach avoids the redundant storage of unchanged elements but only stores the differences between versions. To estimate the resulting storage need we can consider the number of elements in the union of all versions thereby considering elements only once irrespective of their number of occurrence. Since both approaches store the same number of elements $|E|_{v1}$ for the initial version, we further calculate $|E|_n^{naive} / |E|_{v1}$ and $|E|_n^{approach} / |E|_{v1}$ to measure the growth w.r.t. first version v_1.

To determine the influence of ontology changes on the two versioning approaches we consider additions, deletions and modifications of ontology elements. We denote with *add, del* and *mod* the average number of added, deleted and modifies elements, respectively, per version change. Based on the number of elements in the first version $|E|_{v1}$ we estimate the overall number of stored elements for *n* versions as follows.

$$\left|E\right|_n^{naive} = n\left|E\right|_{v1} + \frac{n(n-1)}{2}(add - del) \qquad \left|E\right|_n^{approach} = \left|E\right|_{v1} + (n-1)(add + \mathrm{mod}) \tag{2}$$

The elements stored using the naive versioning technique are n-times the elements of the first version corrected by the elements added or deleted during evolution. Modifications on elements (e.g., attribute value changes) do not affect the overall number of stored elements for the naïve approach. By contrast, our versioning approach stores all elements of the first version plus only the changes (additions, modifications) occurred during evolution. Deletions of elements do not reduce the overall number since the timestamp-based versioning only sets end timestamps for deleted elements, i.e., these elements remain stored.

Since we do not separately store each ontology version, individual versions must be reconstructed utilizing the timestamp information which might be comparatively slow for larger ontologies. Hence we not only evaluate storage requirements but also

Table 1. Typical ontology queries

Q1	Retrieve details (attributes, parents, children) of a given concept
Q2	Retrieve all siblings of a given concept
Q3	Retrieve all concepts in the sub tree of a given concept

Table 2. Storage requirements for $|E|_{v1}$=1000, *add* = 20, *del* = 5, *mod* = 10

n	$\|E\|_n^{usual}$	$\|E\|_n^{approach}$	$\frac{\|E\|_n^{approach}}{\|E\|_n^{usual}}$
10	10,675	1,270	0.12
20	22,850	1,570	0.07
30	36,525	1,870	0.05

the query performance of our versioning approach. For this purpose we analyze query execution times of three typical ontology queries (Table 1). We determine the average execution times, t_Q, for ten executions per query. We also determine the number of result items returned by Q, r_Q, and the average time per result item, t_Q/r_Q.

4.2 Evaluation Scenarios and Results

We first analyze the storage needs of an sample hypothetical ontology and then evaluate our approach on real world data. Table 2 compares the resulting storage requirements for both versioning approaches for a sample ontology of 1,000 elements in the first version and for a different number of versions (n = 10, 20, 30). We further assume average change rates of *add*=20, *del*=5 and *mod*=10 per version. In table shows that the proposed approach has a drastically reduced storage overhead compared to the naive versioning approach. For n versions, our approach requires almost n-times fewer elements than the naive versioning method. Furthermore, our approach performs the better the more versions are included. This is due to the fact that our approach only considers the differences between ontology versions whereas naive versioning redundantly stores ontology elements.

For a real evaluation scenario, we use the *Biological Process* (BP) ontology. The BP ontology is managed by the Gene Ontology source that provides two further ontologies namely the *Molecular Function* (MF) and *Cellular Component* (CC) ontology. These ontologies are typically used to semantically describe properties of gene products or proteins, e.g., in which biological processes they are involved. While changes of these ontologies occur on a daily basis, we restrict ourselves to monthly summarizations between January 2004 and June 2009 (62 versions[1]) in our analysis. The initial version of January 2004 had the smallest number of elements (8,112 concepts, 12,456 relationships, 25,268 attribute values). This number increased by a more than a factor of 3 in the latest version of June 2009 (17,104 concepts, 34,248 relationships, 85,767 attribute values). The following evaluation considers three aspects. We first show a comparison of the storage requirements between our approach and the naive versioning method. We then study the influence of the storage interval (e.g., monthly, quarterly) on our approach. Finally, query performance is analyzed. All analyses for the proposed approach use a MySQL database to store the ontology versions; the overall database size for 62 BP versions is 40 MB. The database runs on a

[1] Note that at the time of this study GO did not provide versions for 4 months in this time period.

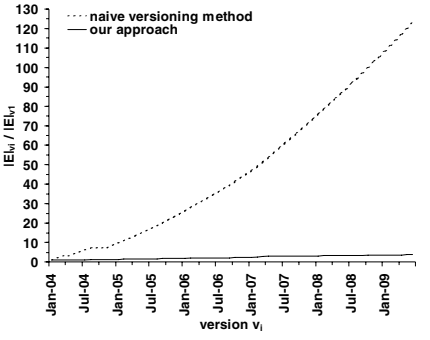

Fig. 3. Comparison of space efficiency of our approach compared to naïve versioning

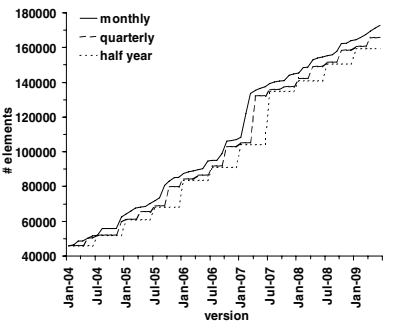

Fig. 4. Comparison of monthly/ quarterly/ semiannually storage requirements

Linux-based 2x Dual-Core 2 GHz Intel machine with 8 GB physical RAM. The tests were performed with disabled database caching in single-user mode.

Fig. 3 illustrates the space efficiency of our approach compared to the naïve versioning method for the considered versioning period (January 2004 - June 2009). The storage requirements (integrated number of elements $|E|_{vi}$) of both approaches are normalized according to the initial version. For both approaches the integration of the initial version takes the same number of elements (about 45,000). For the naïve versioning method, the storage requirements increase substantially with each version and reach 5.6 million elements after the final version corresponding to a growth factor of 124 compared to the first version. By contrast, our approach finally results in only 170,000 stored elements and a very modest growth factor of 3.8 (which corresponds to the overall increase of ontology elements). Given that the naïve versioning needs a 32-fold amount of storage we estimate that the corresponding database would have a size of approx. 1.3 GB compared to the 40 MB of the optimized scheme.

Fig. 4 compares the space requirements for monthly, quarterly and semiannually versioning using our approach. Of course, the space requirements of monthly versioning are in general higher than for quarterly and semiannual versions. However, the difference between the three variants is marginal. For the considered time period until June 2009, monthly storage consumes approx. 170,000 elements compared to about 165,000 and 160,000 elements for quarterly and semiannual versions. The greater accuracy of monthly versions can thus be supported with a minimally increased storage. This underlines the scalability of the proposed versioning approach which is especially valuable when a large number of frequently released ontology versions need to be dealt with.

Finally, we study the query performance of our versioning approach for the three query types of Table 1. For this experiment we use 22 ontology versions from Jan. 2004 to Apr. 2009. The queries are applied to the biological process *behavior* (GO:0007610) that is located on the third level (*biological process* → *response to stimulus* → *behavior*) of the GO-BP ontology. Fig. 5 depicts that the execution times for Q1, Q2, and Q3 are nearly constant over different ontology versions. It takes on average merely 0.17 ms to retrieve all concept details and the siblings of the selected biological process whereas

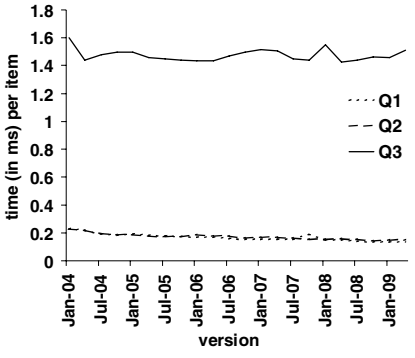

Fig. 5. Query performance analysis, Q1, Q2, Q3 for GO:0007610

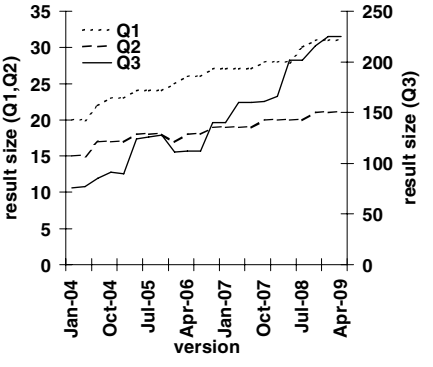

Fig. 6. Number of result items, Q1, Q2, Q3 for GO:0007610

the execution times for querying the complete sub-graph are on average 1.47 ms. This holds despite the increase in the number of stored ontology elements for newer versions. Fig. 6 shows the result sizes of the three queries for the selected biological process which grow for newer versions. The number of concept details (Q1) increases from 20 to 31 between January 2004 and April 2009 (growth factor: 1.55), while the number of concepts in the set of siblings (Q2) and in the sub-graph (Q3) grow by a factors 1.4 and 3, respectively. In summary, our versioning approach is not only very space-efficient but also provides nearly constant execution times of the considered query types.

5 Conclusion and Future Work

We propose a scalable and space-efficient approach to manage the versions for large ontologies. Our ontology versioning model maintains a life time for all ontology concepts, relationships and attribute values. This allows the reconstruction of the valid state of an ontology for any point in time while avoiding the redundant storage of unchanged parts in succeeding ontology versions. We applied our approach to biomedical ontologies but it is not limited to this domain. The evaluated of the storage requirements confirmed the high space efficiency of the proposed approach resulting in up to a n-fold reduction for n versions compared the separate storage of ontology versions. The evaluation of query performance over numerous versions also showed excellent results indicating that the space efficiency does not result in a significant penalization of query execution times.

In future work we plan to extend our approach to the efficient versioning of mappings, e.g., mappings among ontologies (ontology mappings) and mappings associating ontology concepts with objects to semantically describe their properties (annotation mappings). Moreover, the approach can be applied to other domains, e.g. for an evolution analysis of Wikipedia categories.

Acknowledgments. This work is supported by the German Research Foundation, grant RA 497/18-1.

References

1. Berriz, G.F., et al.: Characterizing gene sets with FuncAssociate. Bioinformatics 19(18), 2502–2504 (2003)
2. Day-Richter, J., et al.: OBO-Edit - an ontology editor for biologists. Bioinformatics 23(16), 2198–2200 (2007)
3. Flouris, G., et al.: Ontology change: Classification and survey. Knowledge Engineering Review 23(2), 117–152 (2008)
4. The Gene Ontology Consortium: The Gene Ontology project in 2008. Nucleic Acids Research 36(Database issue), D440–D441(2008)
5. Hartung, M., Kirsten, T., Rahm, E.: Analyzing the Evolution of Life Science Ontologies and Mappings. In: Bairoch, A., Cohen-Boulakia, S., Froidevaux, C. (eds.) DILS 2008. LNCS (LNBI), vol. 5109, pp. 11–27. Springer, Heidelberg (2008)
6. Hartung, M., et al.: OnEX – Exploring changes in life science ontologies. BMC Bioinformatics 10(1), 250 (2009)
7. Klein, M., et al.: Ontoview: Comparing and versioning ontologies. In: Collected Posters of 1st Intl. Semantic Web Conference, ISWC (2002)
8. Klein, M., Fensel, D.: Ontology versioning on the Semantic Web. In: Proc. of the International Semantic Web Working Symposium (SWWS), pp. 75–91 (2001)
9. Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A.: The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 229–243. Springer, Heidelberg (2004)
10. Nadkarni, P.M., et al.: Organization of heterogeneous scientific data using the EAV/CR representation. Journal of American Medical Informatics Association 6(6), 478–493 (1999)
11. Prüfer, K., et al.: FUNC: a package for detecting significant associations between gene sets and ontological annotations. BMC Bioinformatics 8(1), 41 (2007)
12. Sioutos, N., de Coronado, S., Haber, M.W.: NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. Journal of Biomedical Informatics (40), 30–43 (2007)
13. Völkel, M., Groza, T.: SemVersion: RDF-based ontology versioning system. In: Proc. of the IADIS Intl. Conference WWW/Internet, ICWI (2006)
14. Journal of Biomedical Informatics, Special Issue on Auditing of Terminologies 42(3), 402–580 (2009)