Detecting Different Versions of Ontologies in Large Ontology Repositories

Carlo Allocca, Mathieu d'Aquin and Enrico Motta*

Knowledge Media Institute (KMi), The Open University, Walton Hall,
Milton Keynes MK7 6AA, United Kingdom
{c.allocca, m.daquin, e.motta}@open.ac.uk

Abstract. There exist a number of large repositories and search engines collecting ontologies from the web or directly from users. While mechanisms exist to help the authors of these ontologies manage their evolution locally, the links between different versions of the same ontology are often lost when the ontologies are collected by such systems. By inspecting a large collection of ontologies as part of the Watson search engine, we can see that this information is often encoded in the identifier of the ontologies, their URIs, using a variety of conventions and formats. We therefore devise an algorithm, the Ontology Version Detector, which implements a set of rules analyzing and comparing URIs of ontologies to discover versioning relations between ontologies. Through an experiment realized with 7000 ontologies, we show that such a simple and extensible approach actually provides large amounts of useful and relevant results. Indeed, the information derived from this algorithm helps us in understanding how version information is encoded in URIs and how ontologies evolve on the Web, ultimately supporting users in better exploiting the content of large ontology repositories.

1 Introduction

Ontologies are the pillars of the Semantic Web and because more and more ontologies are made available online, finding, understanding and managing online ontologies is becoming more challenging. Indeed, ontologies are not isolated artifacts: they are, explicitly or implicitly, connected to each other [4]. in particular, while being collected by online repositories and search engines, ontologies evolve and different documents can be collected at different times that represent two versions of the same ontology.

A number of studies have intended to tackle some of the challenges raised by ontology versioning, from both theoretical and practical points of view. At the theoretical level, studies have targeted ontology versioning in order to provide a theoretically semantic model for managing ontologies in distributed environments, such as the Web [2,3]. According to [3], the ontology versioning problem

* **This work was funded by the EC IST-FF6-027595 NeOn Project. I would like also to thank Ben Hawkridge for providing very powerful machine to complete the experiments.**

has been defined as *the ability to handle changes in ontologies by creating and managing their own variants/mutants/versions.* In other words, ontology versioning means that there are multiple variants of an ontology around and that these variants should be managed and monitored. Accordingly, tools such as *Evolva* [5] have been developed to support the developers of ontologies in making them evolve and in managing the versions locally. However, such systems use different ways to represent and codify version information, which is not transfered when the ontologies are collected and made accessible through online repositories. Standards such as OWL and OMV [1] include primitives to encode version information as ontology annotations. However, such standards are not universally used and ontology developers rarely make the effort of applying such standards. Instead, they tend to codify information related to the version of an ontology directly in its URI. Indeed, typing the query "*metadata*" currently gives 1356 results in the Watson search engine[1] (valid on the 20/08/2009). However, only inspecting the URIs in the first page of results, we can see that many of these documents (e.g., `http://loki.cae.drexel.edu/~wbs/ontology/2004/01/iso-metadata` and `http://loki.cae.drexel.edu/~wbs/ontology/2003/10/iso-metadata`), represent different versions of the same ontology.

In this paper, we present an algorithm, the *Ontology Version Detector* (OVD) which tries and detect different ways (i.e., different conventions) for encoding version information in ontology URIs in order to derive versioning links between ontologies within a large repository. It relies on a comparison of the URIs of ontologies to detect number differences, which can represent version numbers (e.g., v1.2, v3.6), dates (e.g., 2005/04, 01-12-1999) or other types of versioning information (e.g., time-stamps). One of the advantages of such an approach is that it is based on a set of rules, each encoding a particular pattern for the representation of version information and so, if missing patterns are observed in the collection, they can easily be added and taken into account by the algorithm. In this paper, we detail the set of rules derived from our observations using the Watson ontology search engine.

We conducted an experiment applying OVD on a sub-set of the Watson repository of ontologies containing about 7000 ontologies. While we are aware that the approach implemented by OVD has a number of limitations (i.e., if only looks at the information encoded in the URI through numbers), this experiment showed that a large amount of versioning links implicitly encoded in the URIs of the ontologies can be correctly detected. Indeed, this experiment resulted in 155,589 versioning links, representing 1,365 "evolving ontologies" and which have been evaluated with an estimated precision of 51.2%. In addition, the analysis of these results allows us to identify ways to overcome the limitations of OVD, to better understand how version information is encoded in URIs and to assess how ontologies evolve on the Web, ultimately providing valuable information for the users of ontology repositories.

In the next section, we describe a number of examples and general patterns we observed in the Watson collection of ontologies. Section 3 then details our

[1] http://watson.kmi.open.ac.uk

OVD algorithm for detecting and comparing such patterns. Our experiment on applying OVD is presented in Section 4. Finally, Section 5 discuss the conclusions and future work.

2 Identifying Version Information Patterns

Analyzing a representative sample (nearly 1000) of ontologies from Watson's ontology repository, we have, manually, identified many ontology URIs containing information concerning the version of the ontology. In this paper, we focus on particular versioning patterns. Specifically, we only discuss three classes of URIs that can be compared to establish versioning links between URIs: *Class A*, where the versioning information is encoded in a single number; *Class B*, where the versioning information is expressed by two numbers, which are often the month and year of a date; and *Class C* where the versioning information is expressed by three numbers, which always correspond to complete date.

2.1 Class A: version information expressed by one number

These are simplest and most frequent cases: when the comparison of two URIs would only show a difference in one number. In many examples, this number represents a very simple version number, like in the following example:

Example 1:

1. `http://www.vistology.com/ont/tests/student1.owl`;
2. `http://www.vistology.com/ont/tests/student2.owl`;

However, there can be many variants of such a pattern. In the following example, a time-stamp is used to mark a particular version of the ontology:

Example 2:

```
http://160.45.117.10/semweb/webrdf/#generate_timestamp_1176978024.
  owl
http://160.45.117.10/semweb/webrdf/#generate_timestamp_1178119183.
  owl
```

2.2 Class B: version information expressed by two numbers

Under this category, we find more classical ways to represent version numbers (with a number of the major revision and a number of the minor revision), like in the following example:

Example 3:

1. http://lsdis.cs.uga.edu/projects/semdis/sweto/testbed_v1_1.owl
2. http://lsdis.cs.uga.edu/projects/semdis/sweto/testbed_v1_4.owl

However, a majority of the URIs using two numbers to represent version information use a date format that includes the year and the month. In the following example, the year is the first element to be encoded.

Example 4:

http://loki.cae.drexel.edu/~wbs/ontology/2003/02/iso-metadata
http://loki.cae.drexel.edu/~wbs/ontology/2003/10/iso-metadata
http://loki.cae.drexel.edu/~wbs/ontology/2004/01/iso-metadata
http://loki.cae.drexel.edu/~wbs/ontology/2004/04/iso-metadata

However, there can be four different ways to combine the month and the year: 1) Big endian form (yyyy/mm), in which the year is expressed by four digits; 2) Little endian form (mm/yyyy), in which the year is expressed by four digits; 3) Big endian form yy/mm, in which the year is expressed by only two digits; 4) Big endian form (mm/yy), in which the year is expressed by only two digits. Since we did not encounter examples where the year was expressed with two digits only, we ignore this case in this paper, but rules to cover such patterns can easily be added to the OVD algorithm.

2.3 Class C: version information expressed by three numbers

Under this category we only found cases in which the versioning information is represented through a date format structure using the day, the month and the year. The example below is based on the big endian form (yyy-mm-dd), but as for the cases above, little endian forms (and even middle endian forms, mm-dd-yyy) could be employed.

Example 5

http://ontobroker.semanticweb.org/ontologies/ka2-onto-2000-11-07.daml
http://ontobroker.semanticweb.org/ontologies/ka2-onto-2001-03-04.daml

3 The Ontology Version Detector Algorithm

As shown in the previous section, there is no official and implemented standard or canonical form to represent version information through the URIs. However, the ontology engineers use "semi-rational" criteria to rename new versions, leading to the emergence of common patterns. The goal of the OVD algorithm is to provide a general mechanism to detect such patterns.

3.1 Overview

A general oververview of OVD is described in Figure 1. There are two main steps to the process: the *Selector* component compares URIs to extract sets of numerical differences between them, and the *Recognizer* component detects known patterns in these differences, encoded as a set of rules to generate versioning relations between ontologies. In this paper, we focus on particular versioning patterns. In particular, the ones in which the versioning information is codified by single number or date with two-digit and three-digit format.

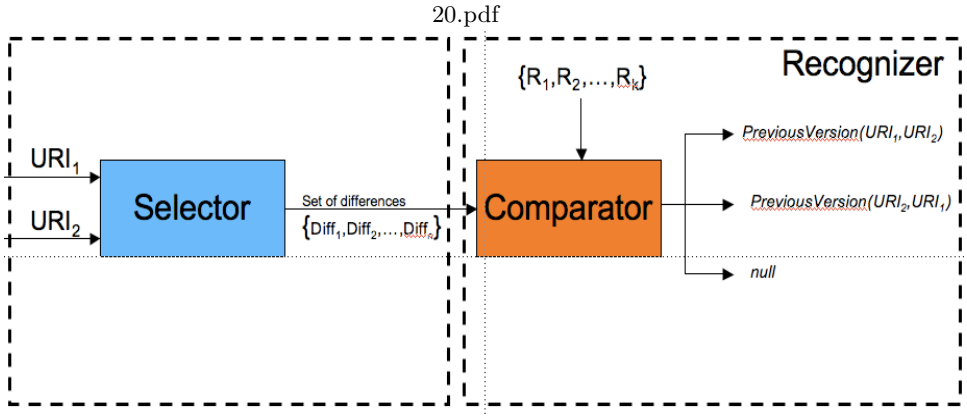

Fig. 1. The main steps of OVD.

The Selector components takes as input a set $\{URI_1, URI_2, ..., URI_n\}$ of URIs of ontologies and returns of set of numerical differences between pairs of URIs. More precisely, to a pair of URIs (URI_i, URI_j), the Selector associate an ordered list of numerical differences represented as pairs (n_{ik}, n_{jk}) where n_{ik} is a number part of URI_i and n_{jk} is a number which replaces n_ik in URI_j. Pairs of URIs for which differences appear in other parts than numbers are simply discarded.

To realize this task, the Selector first *sequences* each URIs in a chain of sections, separating parts that represents numbers from the ones that represent other elements. If two URIs contain the same number of number sections and non-number sections, and if all the non-number sections are equal, the numerical differences are straightforwardly extracted from comparing the number sections of the two URIs with each other. For example, the URIs:

- `http://loki.cae.drexel.edu/~wbs/ontology/2003/10/iso-metadata`
- `http://loki.cae.drexel.edu/~wbs/ontology/2004/01/iso-metadata`

are sequenced in the following way:

- `http://loki.cae.drexel.edu/~wbs/ontology/` ‖ `2003` ‖ `/` ‖ `10` ‖ `/iso-metadata`

– `http://loki.cae.drexel.edu/~wbs/ontology/` ‖ `2004` ‖ `/` ‖ `01` ‖ `/iso-metadata`

Since all the non-number sections are equal and all the number sections are different, the generated differences correspond to the list of pairs of number sections: $[(2003, 2004); (10, 01)]$. It is important to notice here that the number of digits used to represent each number needs to be kept in the representation. Indeed, this information is used as part of the pattern recognition and, for example, '1' should not be considered equivalent to ''01'. We use the notation $|n|$ to indicate the number of digits used in the string representation of the number n in a set of differences.

The second step of the process, the Recognizer component, takes as input the list of differences between pairs of URIs and derive from them versioning relations that should hold between the corresponding ontologies. Versioning relations are represented here as (ordered) pairs of URIs, with $[URI_i, URI_j]$ meaning that URI_j is a more recent version of URI_i. To realize this task, the Recognizer rely on a set of rules that detect specific patterns in the differences, such as the ones identified in Section 2. Below, we detail the set of rules we have defined from our anlysis of the Watson repository.

3.2 Versioning Relation Detection Rules

The *Recognizer* implements a set of rules which are designed to cover the different way that are used to rename a new version, that is, the version information patterns. The rules presented here reflect the main characteristics of the classes of URIs discussed in Section 2, but can be easily extended for additional patterns (many of them being easily derived from the existing rules) but are ignored here as they do not appear in the considered collection of ontologies (e.g., cases where the year is represented with 2 digits, where the date is represented in middle endian form, or the version number is represented with 3 components–x.y.z). Each rule considers a pair of URIs (URI_i, URI_j) and the corresponding list of differences $D_{ij} = [(n_{i1}, n_{j1}), ...]$ to derive a probable versioning relation between URI_i and URI_j.

Class A As indicated before, Class A corresponds to the most straightforward case: there is only one numerical difference between two URIs (i.e. the cardinality $|D_{ij}|$ of D_{ij} is 1). In this case, we only need to compare the number in question to derive which version came first.

R1 IF $(|D_{ij}| = 1)$ THEN
  IF $(n_{i1} < n_{j1})$ THEN
    $[URI_i, URI_j]$
  ELSE
    $[URI_j, URI_i]$

The rule R1 addresses the cases such as the ones shown in Example 1 and Example 2.

Class B Class B is a more complicated example, as it corresponds to the cases where two numbers differ from one URI to the other. Therefore, to realize an appropriate comparison, it is first needed to find which number is the most significant. We distinguish two main cases: 1- the version information corresponds to a version number, in which case the number on the left is more significant, or 2- the version information corresponds to a date including the year and month, in which case, the year is more significant.

Therefore, in order to distinguish these different situations, we need to be able to recognize and year and a month from any other number. We define the following 2 predicates, $year(n)$ and $month(n)$, with return true if the number n can be a year or a month respectively:

$$year(n) :: |n| = 4 \text{ and } 1995 \leq n \leq current\_year$$
$$month(n) :: |n| = 2 \text{ and } 01 \leq n \leq 12$$

These conditions assume that ontologies can only have been created from 1995 to the present year, that months are always represented with 2 digits and years with 4 digits. While these assumptions might appear restrictive, they reflect our observations on the Watson collection of ontologies and help in avoiding unnecessary noise.

Based on $year(n)$ and $month(n)$, we can derive the following predicates that indicate if 2 numbers, n_1 and n_2 can represent a date, either in big endian or in little endian forms:

$$dateLE(n_1, n_2) :: month(n_1) \text{ and } year(n_2)$$
$$dateBE(n_1, n_2) :: year(n_1) \text{ and } month(n_2)$$

Finally, using these conditions, we can define the three following rules: R2 for cases where 2 numbers differ but don't correspond to a date (in which case the number on the left is assumed to be the most significant), R3 for cases where a date in little endian form is used, and R4 for cases where a date in big endian form is used.

R2 IF ($|D_{ij}| = 2$) THEN
IF (NOT (dateLE(n_{i1}, n_{i2}) AND dateLE(n_{j1}, n_{j2}))
AND NOT (dateBE(n_{i1}, n_{i2}) AND dateBE(n_{j1}, n_{j2}))) THEN
IF ($n_{i1} = n_{j1}$) THEN
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

R3 IF ($|D_{ij}| = 2$) THEN
IF (dateLE(n_{i1}, n_{i2}) AND dateLE(n_{j1}, n_{j2}))
IF ($n_{i2} = n_{j2}$) THEN
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

R4 IF ($|D_{ij}| = 2$) THEN
IF (dateBE(n_{i1}, n_{i2}) AND dateBE(n_{j1}, n_{j2}))
IF ($n_{i1} = n_{j1}$) THEN
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

Class C Finally, Class C corresponds to the cases where 3 numerical differences exist between the considered URIs. As for class B, it is important in that case to first detect which is the most significant of these numbers. However, we haven't encountered examples other than the representations of dates using 3 numbers. Therefore, we only define the rules corresponding the dates, either in big endian or in little indian form. We define a new predicate, $day(n)$ which indicates if a number could be a day of a month:

$$day(n) :: 01 \leq n \leq 31$$

as well as the conditions to recognize dates with 3 numbers

$$dateLE(n_1, n_2, n_3) :: day(n_1) \text{ and } month(n_2) \text{ and } year(n_3)$$
$$dateBE(n_1, n_2, n_3) :: year(n_1) \text{ and } month(n_2) \text{ and } day(n_3)$$

Rules R5 and R6 corresponds to the two cases of dates with 3 numbers, in little endian and big endian forms respectively.

R5 IF ($|D_{ij}| = 3$) THEN
IF (dateLE(n_{i1}, n_{i2}, n_{i3}) AND dateLE(n_{j1}, n_{j2}, n_{j3}))

IF $(n_{i3} = n_{j3})$ THEN
IF $(n_{i2} = n_{j2})$ THEN
IF $(n_{i1} < n_{j1})$ THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF $(n_{i2} < n_{j2})$ THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF $(n_{i1} < n_{j1})$ THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

R6 IF $(|D_{ij}| = 3)$ THEN
IF (dateBE(n_{i1}, n_{i2}, n_{i3}) AND dateBE(n_{j1}, n_{j2}, n_{j3}))
IF $(n_{i1} = n_{j1})$ THEN
IF $(n_{i2} = n_{j2})$ THEN
IF $(n_{i3} < n_{j3})$ THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF $(n_{i2} < n_{j2})$ THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF $(n_{i1} < n_{j1})$ THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

4 Experiment and Evaluation

OVD is based on particular patterns which have been identified by manually analyzing Watson's repository and are not formally specified. Consequently, an empirical evaluation is crucial to verify the correctness (i.e., the precision) of OVD. In addition, such an evaluation gives us an insight on the way URIs are effectively used to encode version information, and on how we could improve OVD.

4.1 Experiment data

Here, we experiment with 4 sets of ontologies of increasing sizes extracted from the Watson collection (see Table 1). Queries 1 to 3 correspond to the OWL ontologies returned with the queries "student", "man", and "person" respectively. Query 4 corresponds to a set of nearly 7000 OWL ontologies not corresponding to any particular query.

QueryName	QueryPhrase	Number of Ontologies
Query1	Student	90
Query2	Man	115
Query3	Person	875
Query4	*	6898

Table 1. Queries and corresponding ontology collections.

We ran the OVD algorithm on these 4 sets of ontologies to discover versioning links between the ontologies they contain, separating the results from the different rules, in order to allow evaluating each rule separately[2]. Table 2 shows the number of pairs of ontologies which OVD has detected a versions of each other (see 3rd column) for each set of ontologies and each rules. In total, 155,589 pairs of ontology potential ontology versions have been detected. As can be seen, patterns corresponding to R1 and R2 seem to be the most applied to represent version information, with a clear different for R2 in the case of Query 4. We can also remark that R3 and R5 were never triggered in our datasets. One corresponds to dates of the form mm/yyyy and the other, to dates of the form yyyy/mm/dd.

4.2 Computing chains of ontology versions

While individual versioning relations are interesting to consider, many of these relations are parts of sequences of successive versions. We define and compute such *chains* of ontologies as the connected paths in the graph formed by the versioning relations detected by OVD. More formally,

Definition 1 *Given set of pairs of ontologies* $\{(O_i, O_j)\}$*), an ontology chain is defined as a sequence of ontologies,* $O_1, O_2, ..., O_{n-1}, O_n$*, such that* O_i *is the previous version of* O_{i+1}*.*

For example, using the abbreviation A for `http://oaei.ontologymatching.org/2004/Contest/` we can compute the chain of ontology versions: [`A/testbed_v1_1.owl`, `A/testbed_v1_3.owl`, `A/testbed_v1_4.owl`]

The 4th column of Table 2 shows for each rule in each dataset the number of chains of ontologies that can be computed from the result of OVD.

[2] It is worth mentioning that this computation took under 5 minutes on a Mac Laptop

QueryName	Rule	Detected Pairs	Number Of Chains
Query1	R1	16	5
	R2	0	0
	R3/5	0	0
	R4	0	0
	R6	0	0
Query2	R1	16	6
	R2	0	0
	R3/5	0	0
	R4	11	5
	R6	0	0
Query3	R1	41	10
	R2	4	3
	R3/5	0	0
	R4	10	2
	R6	0	0
Query4	R1	17334	511
	R2	138119	843
	R3/5	0	0
	R4	38	7
	R6	10	4

Table 2. Results of running OVD on sub-sets of the Watson collection of ontologies.

4.3 Evaluation

In order to evaluate the results of OVD, we manually checked the correctness of the chains obtained for each rule in each of our datasets. Here, we assume that if one of the versioning relation in a chain is incorrect, the entire chain is incorrect. For the dataset corresponding to Query 4, we only evaluated a sample of 10 chains in the case of R1 and 18 chains in case of R2. We chose these samples randomly, but trying to get examples coming from different sites in order to evaluate different conventions.The third and fourth columns of Table 4 show the result of this evaluation.

Using the usual measure of precision, we can evaluate the overall performance of the OVD algorithm (51.2%), as well as the individual performance of each rule. The results are presented in Table 4. Looking at the incorrect results, we can draw a number of conclusions concerning the way we can improve OVD. Indeed, for example, it can seen that some of the rules provide more accurate information than others. Also, it can be seen that longer chains tend to be incorrect. On of the reasons is that many incorrect results come from automatically generated ontologies, which uses numbers not to represent version information, but record numbers. Using such information, we can compute different levels of confidence for the results. Finally, successive versions of ontologies tend to be similar to each other. Using a measure of similarity can help us in sorting out the correct

Query	Rule	Correct Chains	Incorrect Chains	Average Length	Max Length
Query 1	R1	3	2	2.6	3
	R2	0	0	0	0
	R4	0	0	0	0
	R6	0	0	0	0
Query 2	R1	4	2	2.3	3
	R2	0	0	0	0
	R4	4	1	2.6	3
	R6	0	0	0	0
Query 3	R1	6	4	1.8	3
	R2	3	0	2.3	3
	R4	0	2	2.5	3
	R6	0	0	0	0
Query 4	R1	5	5	89.05	230
	R2	9	9	71.13	154
	R4	4	3	2.2	3
	R6	2	2	2.75	3

Table 3. Result of the evaluation of OVD

	Total	R1	R2	R4	R6
Precision	51.2%	50%	50%	57%	50%

Table 4. Precision OVD's results.

results from the incorrect ones. Finally, some incorrect results come from RDF documents describing dated events (e.g., ESWC2006 and ESWC2007). Checking if the ontology describes dated element can also give indication that the result should be seen as incorrect.

5 Conclusion and Future work

In this paper, general patterns which convey ontology version information directly into their URIs have been investigated, in the context of large ontology repositories such as Watson. In particular, we have identified 6 patterns which have been formalized by 6 rules. Informally, those rules describe regular sequence of characters discernible as part of the URI which hold the version information and can be used to derive versioning relations between ontologies.

Based on these rules, we described and designed OVD (Ontology Versioning Detector), which is an algorithm for detecting different versions of ontologies in large ontology repositories. It is based on two main steps: the *Selector* which compares URIs to extract sets of numerical differences between them, and the *Recognizer* which identifies the well-known pattern and try to figure out which

ontology comes first. OVD has been evaluated over Watson's ontology collection, providing useful and relevant results. Indeed, the information derived from this algorithm helps us to understand how the versioning information is encoded in URIs and how ontologies evolve on the Web, ultimately supporting users in better exploiting the content of large ontology repositories. The current implementation of OVD detects different versions of the ontologies when the versioning information is expressed by single number or date-pattern. In other case OVD does not detect. But, we want to extend this work considering different directions. First, new patterns not exclusively based on numbers could be detected, such as "October-2006"; new patters based on more than four numbers. Second, according to [3] the modification of an ontology can lead to a new version which is completely different from the original one. Although in practice, by analyzing Watson's ontology repository, we can see that it is very likely fro 2 versions of the same ontology to be similar. We can use such information to increase the precision of OVD. Finally, we can also exploit explicit version information encoded using OWL primitives and consider the overlap with information encoded in URIs.

References

1. J. Hartmann, R. Palma, and et al. Ontology metadata vocabulary and applications. pages 906–915, OCT 2005.
2. Jeff Heflin and James A. Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 443–449. AAAI Press / The MIT Press, 2000.
3. M. Klein and D. Fensel. Ontology versioning on the semantic web. *Proc. of the Inter. Semantic Web Working Symposium (SWWS)*, pages 75–91, 2001.
4. A. Kleshchev and I. Artemjeva. An analysis of some relations among domain ontologies. *Int. Journal on Inf. Theories and Appl*, 12:85–93, 2005.
5. F. Zablith. Evolva: A comprehensive approach to ontology evolution. *In: Proceedings of 6th European Semantic Web Conference (ESWC) PhD Symposium LNCS 5554. eds. L. Aroyo et al., Springer*, pages 944–948, 2009.