SKOS-Utils: Developing and Checking SKOS Knowledge Graphs (Tool Presentation)

Joachim Baumeister[1,2,*], Valentin Roß[1,2]

[1]University of Würzburg, Germany
[2]denkbares GmbH, Germany

Abstract

We introduce the tool suite SKOS-Utils for the development and quality assessment of SKOS vocabularies (Simple Knowledge Organisation System). SKOS is a wide-spread standard for organizing hierarchical knowledge structures, that uses RDF as basic data model. SKOS-Utils provides converters for spreadsheets and mindmaps in order to facilitate the bottom-up development of SKOS vocabularies. A suite of quality checks is used for testing the created vocabularies.

Keywords

knowledge engineering, ontologies, mindmaps, quality assessment, evaluation, knowledge bases

1. Introduction

By now, SKOS (Simple Knowledge Organization System) is a well-accepted W3C standard for defining (hierarchical) knowledge structures based on the RDF model [1]. Many vendors support the definition, storage, and use of SKOS schemes. For example, industrial applications implement the SKOS standard for the representation and interchange of master data, e.g., component structures, functional schemes, and parts list hierarchies.

The wide use of SKOS is justified by the simplicity of the underlying data model. The center of the SKOS data model defines `skos:Concept` to represent an element in the knowledge structure. Instances of `skos:Concept` are organized in a hierarchy by defining parent/child relations. In SKOS, these relations are generally named `skos:broader` and `skos:narrower` for a parent and child relationship, respectively. Figure 1 depicts an example of the SKOS concept `Frame` having a number of labels defined by using the standard SKOS properties. For industrial applications, these relations can be subclassed as for instance `subComponentOf`, `partOf`, and `hasFunction`.

Besides the hierarchical relationship in the knowledge organization structure, the labeling of the concepts is a central feature of SKOS. Lexical labels with different semantics are introduced: The property `skos:prefLabel` defines the relation for the preferred label of the concept, whereas uses of `skos:altLabel` point to alternative lexical labels of the concept. As a third property `skos:hiddenLabel` stores lexical labels of the concept, that are usually not shown to the user, e.g., typical misspellings and internal identifiers. The core standard of SKOS defines the properties shown above. With the extension SKOS-XL (eXtension for Labels), the properties

FGWM 2023: German SIG on Knowledge Management
*Corresponding author.
joba@uni-wuerzburg.de (J. Baumeister)

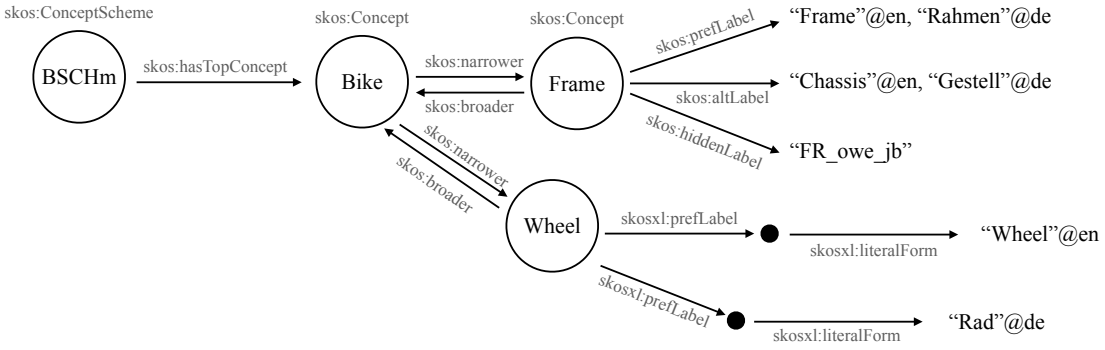

Figure 1: Example of SKOS concepts with preferred, alternative, and hidden labels by using standard SKOS and SKOS-XL.

are expanded so that concepts can point to first class instances of lexical entities [1]. Then, a label can store additional information besides its literal form, e.g., relations to other labels and (domain specific) provenance information. Figure 1 shows the use of SKOS-XL preferred labels for the concept `Wheel`.

Concepts are grouped in a `skos:ConceptScheme` by using the `skos:inScheme` property. The entry into a concept scheme is represented by using the property `skos:hasTopConcept`, which points to the topmost concepts of this scheme. See the ConceptScheme instance `BSCHm` in Figure 1. Please note, that there is no constraint, that only one top concept must exist.

When not directly maintained in a SKOS data model but in a proprietary system, the data is often converted to SKOS representation for interchange in the IT landscape. For instance, master data located originally in an ERP system is exported as SKOS data for the interchange with a knowledge graph application. But also for the creation of new structures the SKOS standard has shown to be beneficial. Here, the concepts of the structure are defined in standard applications like mindmapping tools or even spreadsheet editors. After the initial creation phase, these structures are later transferred to a graph database or ERP system. With these use cases we experienced, that SKOS data is often defined without tools checking for validity or general quality.

1.1. Contribution

This paper introduces the modular open-source tool suite SKOS-Utils. The suite provides a collection of Python scripts that help during the development of SKOS vocabularies and the continuous quality assessment of the resulting knowledge graphs.

There exist various applications tailored to the development of SKOS vocabularies. While these tools offer a wide range of functionalities and benefits for professional authors, the tools lack intuitivity needed for the first steps of developing a new vocabulary, especially for domain specialists not familiar with the engineering of ontologies. We experienced the use of already known tools more accessible for domain specialists, e.g., using spreadsheet editors and mindmapping tools. That way, we claim that it is more productive to start with such simple

applications for the initial structure before switching to tailored tools for supporting the life cycle of the created vocabulary.

1.2. Related Work

Quality Assessment As a primary document besides the W3C standards, Baker et al. [2] introduce the main components of SKOS and the design decisions made. They also state an initial set of *integrity conditions* for SKOS data. Mader et al. [3] present a collection of quality issues that can occur in SKOS vocabularies. The structured approach is implemented in the tool qSKOS and they describe the analysis of many openly available vocabularies, such as DBpedia, AGROVOC, and MeSH. Suominen and Mader [4] extend this approach by also providing correction heuristics for a part of the presented anomalies.

Development Busse [5] introduces an approach to transform FreeMind Mindmaps into RDF ontologies on a syntactic level by using the XSLT scripting language. Lacasta et al. [6] describe a tool for the development and visualization of SKOS data especially focussed on digital libraries. While having a strong support for typical librarian tasks the tool is build mainly as a monolithic application.

2. Methods

The suite SKOS-Utils offers a collection of scripts for developing and checking SKOS data. Due to the *intrinsic nature of incompleteness*[1], the suite is designed to be easily extended. We distinguish the parts *development* and *checking*, whereas each part offers a variety of (extensible) modules for the specific task.

An extensive discussion of methods for checking vocabularies can be found in [3, 4]. We describe the particular modules in the following in more detail. Please note, that the suite is under continuous development and thus we can only describe the current state of implementation.

2.1. Development

The development part provides the standardized SKOS RDF data model as recommended by the W3C [1]. We can simply switch between the well-known RDF syntax definitions XML, Turtle, and JSON-LD. The suite provides scripts to bridge this data model from/to the mindmapping application XMind[2], to Graphviz[3], and from/to the spreadsheet application MS-Excel[4], see Figure 2. That way, we can develop SKOS structures using this tools in a roundtrip engineering approach. XMind and Excel, respectively, are popular for creating (hierarchical) structures in industry. For quick visualization purposes, also an ASCII export is provided, that prints basic information of the SKOS hierarchy as a dashed-tree into a text file.

[1] Here, we follow the general rule that every new project will have new requirements and additions to existing functionality.
[2] https://xmind.app
[3] https://graphviz.org
[4] https://www.microsoft.com/microsoft-365/excel

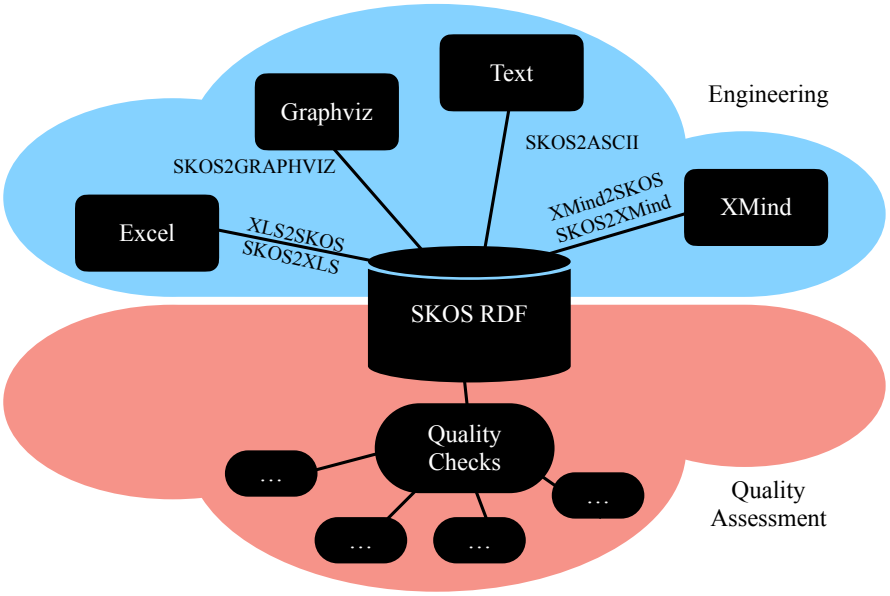

Figure 2: Overview of SKOS-Utils with scripts for the development and the quality assessment of SKOS data.

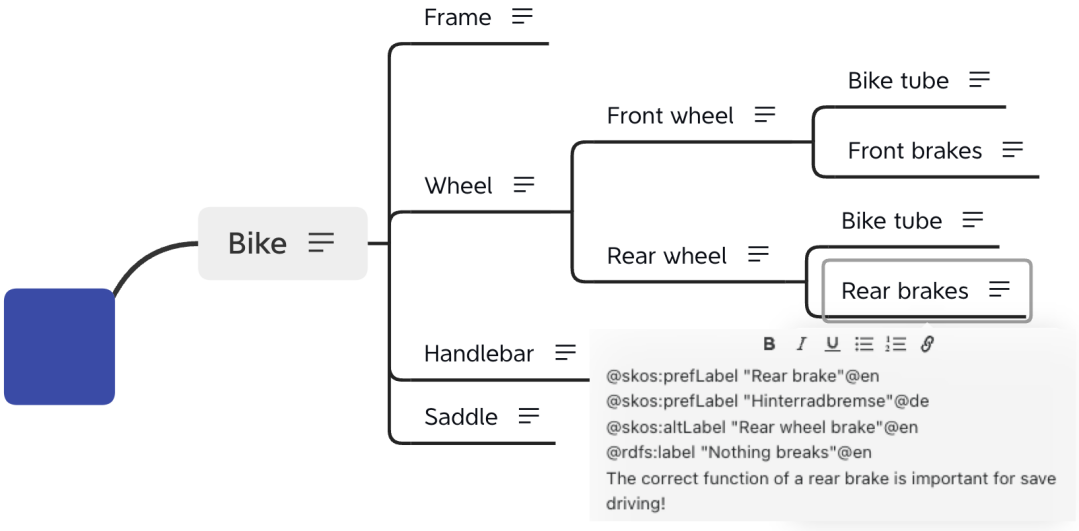

Figure 3: Example of a XMind structure with SKOS properties defined as notes of hierarchy nodes.

Mindmaps In Figure 3 we see an example XMind structure of an exemplary bike definition. Basically, mindmaps allow for the definition of nodes and their hierarchical relationship (general parent-children). This relation is mapped to the `skos:broader` and `skos:narrower` properties, respectively. Further common relations in SKOS are represented by using the 'note' attribute of nodes. When one wants to define preferred labels (`skos:prefLabel`) or alternative labels

(skos:altLabel), these are written into the note. By convention, the user defines those properties in separate lines, where each line needs to start with a @ character. All lines not starting with a @ are handled as standard comments and are transferred into a skos:note property. In the example in Figure 3 we see the note window for the node Rear brakes.

When we apply the script XMind2SKOS of SKOS-Utils a SKOS representation in RDF is generated. The following RDF in Listing 1 is an excerpt for the node Rear brakes (here in turtle syntax). Please note, that unique URIs are generated for each concept since we cannot rely on the unique names in the mindmap. The inverse script SKOS2Xmind converts RDF data to an XMind data format.

Listing 1: Sample output of a SKOS concept.

```
ex:C_5b3154 a skos:Concept ;
    skos:prefLabel "Hinterradbremse"@de,
        "Rear brake"@en ;
    skos:broader ex:C_02b4cc ;
    skos:inScheme ex:Functions ;
    skos:note "@order: 10",
        "@uuid: C_5b3154",
        "@xmind_name: \"Rear brakes\"",
        "The correct function of a rear brake is important for save
           driving!" ;
    rdfs:label "Brake"@en ;
    skos:altLabel "Rear wheel brake"@en .
```

Graphviz SKOS-Utils provides a converter to the dot format of Graphviz. By default, the converter prints the SKOS concepts as nodes. The node label is defined by the preferred label of a specified language. Furthermore, the class type and the preferred labels can be printed in the node. Also, you can define which properties should be drawn in the graph, additionally. Graphviz provides different layout strategies and output formats, such as PDF, PNG, and SVG. Figure 4 depicts the example vocabulary rendered by Graphviz.

Spreadsheets The bridge modules SKOS2XLS and XLS2SKOS of SKOS-Utils transfer SKOS RDF to and from MS-Excel spreadsheets. In the spreadsheet representation, each SKOS concept is located in a separate row, whereas the SKOS properties are represented by the respective columns. Additional properties denoted by @ are exported as single columns. In Figure 5, a MS-Excel sheet is depicted, which was generated from the RDF file and the XMind structure in Figure 3, respectively. The hierarchy is represented in *level notation*, i.e., column Level denotes the hierarchical position of the SKOS concept. A concept C_i with a level i describes the child of a concept C_{i-1} when the parent C_{i-1} occurs immediately before concept C_i. For example, Figure 5 shows the levels in Column A, where concepts Frame and Wheel are children of Bike. This kind of hierarchy level definition is common for exports/imports into ERP systems such as SAP.

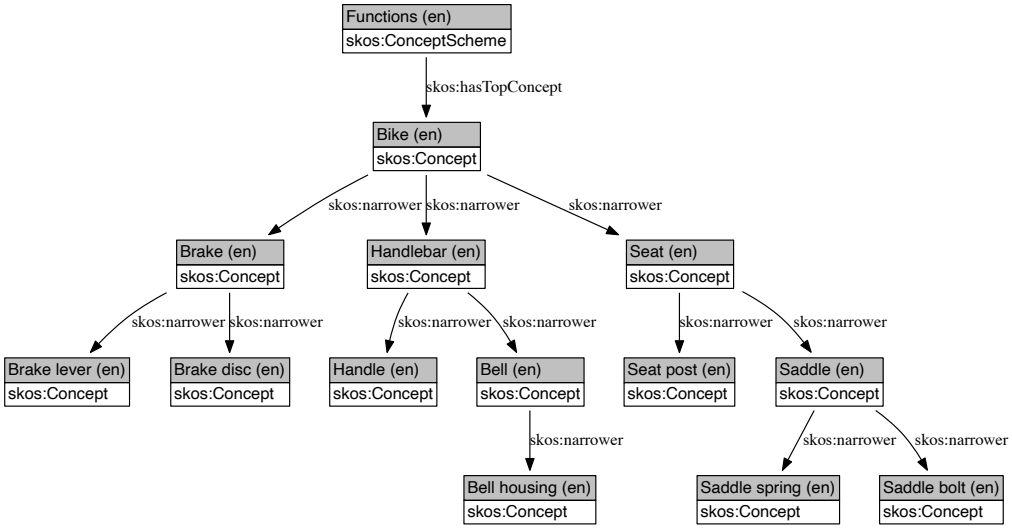

Figure 4: Graphviz layout as PDF output.

	A	B	C	D	E	F	G	H
1	**Level**	**URI**	**@uuid**	**@xmind_name**	**skos:prefLabel**	**skos:prefLabel@de**	**skos:prefLabel@en**	**skos:altLabel@en**
2	1	ex:C_522022	C_522022	Root	Root	Root		
3	2	ex:C_d96111	C_d96111	Bike		Fahrrad	Bike	Bicycle
4	3	ex:C_4df42b	C_4df42b	Frame		Rahmen	Frame	
5	3	ex:C_211fc9	C_211fc9	Wheel		Rad	Wheel	
6	4	ex:C_d354cf	C_d354cf	Front wheel		Vorderrad	Front wheel	
7	5	ex:C_e10fd7	C_e10fd7	Bike tube		Fahrradschlauch	Bike tube	
8	5	ex:C_8f313f	C_8f313f	Front brakes		Vorderradbremse	Front brake	
9	4	ex:C_288bce	C_288bce	Rear wheel		Hinterrad	Rear wheel	
10	5	ex:C_44ed11	C_44ed11	Bike tube		Fahrradschlauch	Bike tube	
11	5	ex:C_bbf6dd	C_bbf6dd	Rear brakes		Hinterradbremse	Rear brake	Rear wheel brake
12	3	ex:C_8dad1a	C_8dad1a	Handlebar		Lenker	Handlebar	
13	4	ex:C_b1d6b6	C_b1d6b6	Bell		Klingel	Bell	
14	3	ex:C_e69ec8	C_e69ec8	Saddle		Sattel	Saddle	

Figure 5: Example of an Excel sheet generated from the SKOS RDF structure above.

2.2. Quality Assessment

For the quality assessment a suite of configurable checks is executed as a pipeline. The results for each check are collected and reported after finishing the entire pipeline. We provide two alternative interfaces for the implementation of checks, i.e., as SPARQL queries [7] and as Python classes. We first introduce the definition of a quality check as a SPARQL query and then show an exemplary implementation of a check in Python.

Check Definition The current state of SKOS-Utils brings a number of checks, that are usable for the quality assessment of SKOS data. The extension of the suite is under development.

We provide two interfaces to implement project specific checks: 1) New checks basically defined by a SPARQL query are implemented by extending the python class `StructureTestInterfaceSPARQL`. 2) Checks navigating the graph object structure are implemented by extending the python class `StructureTestInterfaceNavigate`.

Listing 2 shows an example of the SPARQL approach and implements the check `SinglePrefLabelCheckerSPARQL`. It looks for deficient SKOS concepts that have more than one preferred label for the same language. For the sake of simplicity we omitted the query for retrieving SKOS-XL prefLabel occurrences, that is also part of the actual implementation.

Listing 2: Check definition *SinglePrefLabelChecker* implemented as SPARQL query.

```
class SinglePrefLabelCheckerSPARQL(StructureTestInterfaceSPARQL):
    """
    Check whether every concept has at most one prefLabel for
    each language. Considers prefLabels defined in standard SKOS.
    """
    @property
    def status(self):
        return "Error"

    def message(self, result_df):
        ...

    @property
    def query(self):
        return """
            SELECT DISTINCT ?concept
            WHERE { { ?concept skos:prefLabel ?label1, ?label2 . }
                    FILTER ( lang(?label1) = lang(?label2) &&
                             ?label1 != ?label2 )
            }
            GROUP BY ?concept
            """
```

A check is basically defined by overwriting three methods: The method `status` signifies the escalation level in the case of a found deficiency. The levels *info*, *warning*, and *error* are possible. The method `message` defines the information given to the user in case of found deficiencies. The actual logic for finding deficiencies is specified in the method `query`. Here, a SPARQL query for finding the wanted deficiencies needs to be given. In the standard case, the check does not report an error/warning, when the query returns no results. All results returned by the query, in turn, are interpreted as deficiencies.

Listing 3 shows the same method as native implementation. Native implementations navigate over the graph data model and try to find the deficiencies programmatically. By interfacing the class `StructureTestInterfaceNavigate` we usually only need to implement the method `find_concepts()` with the logic for finding the corresponding deficient concepts.

Listing 3: Check definition *SinglePrefLabelChecker* implemented by navigating the graph object model.

```
class SinglePrefLabelChecker(StructureTestInterfaceNavigate):
    @property
    def status(self):
        ...

    def message(self, result):
        ...

    def find_concepts(self, graph):
        bad_concepts_list = []
        for concept, p, o in graph.triples((None, RDF.type, SKOS.
           Concept)):
            labels = self.all_pref_labels(concept, graph)
            if self.duplicate_labels(labels):
                bad_concepts_list.append(concept)
        return bad_concepts_list
```

Implementation note: Checks defined by SPARQL queries come with the advantage of easy definition. Additionally, SPARQL is a well-accepted W3C standard that is widely used. That way, SPARQL allows for a quick and declarative addition of new checks. However, the disadvantage of that approach is that the execution performance of SPARQL queries is significantly worse than an implementation working directly on the graph object model.

Pipeline Definition The quality assessment of SKOS data is performed by defining a pipeline of quality checks. A configuration file defines the pipeline in YAML syntax. The pipeline execution script `SKOSQualityChecker` performs the actual assessment of the SKOS data using this configuration file. Listing 4 shows an exemplary configuration for the quality assessment:

Listing 4: Sample definition of a check pipeline configuration.

```
input: tests/Testdata/testfile.rdf
output: test_results
add_datetime_to_output: True
write_results_to_excel: True
tests:
   - LooseConceptIdentifier
   - SchemeCoherenceChecker
   - SinglePrefLabelChecker
   - ...
```

The YAML configuration describes the SKOS RDF file under analysis (`input`), the name of the results file (`output`). In the section (`tests`) the actual pipeline of tests is defined. Further, we can toggle whether we want to write a result output and whether we want to log the date/times of the tests.

That way, quality assessments are easily executable the headless way, for example, as part of a nightly continuous integration step. The results of an assessment step can be persisted into an Excel file for subsequent analysis.

3. Case Study

SKOS-Utils is implemented as a Python script collection. For the access to RDF vocabularies we use the popular library rdflib[5]. For the access to Excel files we use the pandas framework[6].

In the following we report the experiences we made during the use of the development scripts in an industrial project. We further demonstrate the implementation of the quality assessment part by reporting the runtime and findings for a collection of known SKOS vocabularies.

3.1. Development with SKOS-Utils

In the context of a customer project we started the development of a functional structure of machines in 2022. The functional structure was created on the green field initialized by a number of workshops with domain specialists from different areas, e.g., engineering, safety, and after sales.

Due to the collaborative set-up of the workshop the use of a mindmapping improved the intuitive creation and change of the structure. We used the bridge to the spreadsheet application for subsequent and iterative analyses by the domain specialists in their own areas. These analyses mainly covered the alignment of existing structures to the newly developed structure. Here, a spreadsheet effectively helped the alignment process. The development of the functional structure is still in progress.

3.2. Quality Assessment of SKOS Vocabularies

Data Sets We selected a collection of popular SKOS vocabularies in order to demonstrate the performance and usability of the developed SKOS-Utils. The *DemoBike* vocabulary is a self-created SKOS structure describing an exemplary bicycle configuration. Here, example defects are included to test the functionality of the SKOS-Utils implementation. The *EuroVoc*[7] vocabulary is a multi-lingual thesaurus created and managed by the European Union publications office and describes terms in languages of EU countries and EU-candidate countries. The *AGROVOC* vocabulary is developed and coordinated by the Food and Agriculture Organization (FAO) of the United Nations [8]. As stated at its homepage "AGROVOC (is) a valuable tool for data to be classified homogeneously, facilitating interoperability and reuse." [8] The UNESCO Thesaurus[9] (UNESCO THES) defines terms in the domain of education, culture, natural sciences, social and human science. The terms are used as metadata for document annotation.

Results We used the introduced data sets with the check implementations of SKOS-Utils. The aim of this experiments was to measure the application runtime and the general functionality of the implementation. For the experiments we used a Macbook Pro M1 Max with 32GB RAM (Python 3.11). Table 2 shows runtime numbers performing the particular checks. For each check the runtime of the navigate implementation is shown and the corresponding SPARQL

[5] rdflib: https://rdflib.readthedocs.io
[6] pandas: https://pandas.pydata.org
[7] EuroVoc: https://op.europa.eu/s/yTkE
[8] AGROVOC: https://www.fao.org/agrovoc/
[9] UNESCO THES: http://vocabularies.unesco.org/thesaurus

Table 1
Quantities of the used SKOS vocabularies

Name	#Schema	#Concepts	#prefLabels	#prefLabels XL	#Relations	#Languages
BikeDemo (2023-07)	1	13	33	3	118	4
EuroVoc (v4.17)	129	7,403	202,008	411,242	3,662,023	30
AGROVOC (2023-06-06)	1	40,983	0	986,077	5,920,997	58
UNESCO THES (v2)	1	4,408	17,980	0	75,202	4

Table 2
Runtimes of implemented checks in *sec:ms* (*sec* omitted when faster).

Name	BikeDemo		EuroVoc		AGROVOC		UNESCO THES	
SinglePrefLabel	002 / -	2	01:493 / -	0	04:585 / -	0	062 / -	0
SchemeCoherence	001 / -	1	00:171 / -	345	00:583 / -	0	073 / -	0
SchemeIntegrity	001 / 073	1	00:043 / 00:482	0	00:212 / 01:961	0	023 / 319	1
TopConceptIdentifier	001 / -	5	00:003 / -	5	00:002 / -	25	035 / 416	1
IncompleteLanguageCoverage	023 / -	1	03:238/ -	7,403	09:952 / -	40,983	123 / -	35
LabelConflict	001 / -	2	00:688 / -	218	00:190 / -	0	088 / -	10
OrphanConcept	001 / 011	1	00:209 / 01:042	0	00:803 / 04:150	0	125 / 679	0
OmittedTopConcepts	001 / -	3	00:064 / -	0	00:250 / -	0	033 / -	0
SolelyTransitivelyRelatedCon	001 / 004	2	00:027 / 00:050	0	00:156 / 00:262	0	016 / 031	0
TopConHavingBroaderCon	001 / 006	1	00:006 / 00:087	0	00:001 / 00:007	0	004 / 046	0
CyclicHierarchicalRelation	001 / -	0	00:269 / -	56	00:760 / -	533	131 / -	0
ValuelessAssociativeRelations	001 / -	4	00:181 / -	221	00:115 / -	1,362	185 / -	1,474
InvalidLanguageTag	001 / -	1	02:306 / -	0	08:704/ -	0	117 / -	0

implementation, when available ("–" otherwise). The runtime numbers in Table 2 are formatted as sec:ms, whereas *sec* is omitted, when the check only had a milliseconds runtime. After the runtimes the count of found anomalies is given.

We see that accessing the SKOS graph by using SPARQL is significantly larger than accessing the graph via the navigation features of the library. This observation is also made for other programming libraries and therefore not surprising.

When looking at the findings in the table, the BikeDemo vocabulary shows findings for all checks. This ist not surprising, since the knowledge base is used as test for all checks, The vocabulary AGROVOC has a high number of *incomplete language coverage*: The vocabulary defines in total 58 languages but mostly uses for the single concepts only about 20 languages. For the UNESCO thesaurus 35 concepts with incomplete language coverage are detected, e.g., not all concepts define preferred labels for Russian language.

4. Conclusions

We introduced the tool suite SKOS-Utils as a collection of useful Python scripts for the development and quality assessment of SKOS vocabularies. SKOS became one industrial standard for creating and interchanging knowledge organization structures. Due to the heterogenous infrastructures of existing IT landscapes in industries, a flexible and open tool suite is necessary to interweave SKOS vocabularies into existing applications.

The described SKOS-Utils is at the beginning of its implementation. While the available

scripts have proven to be useful in daily development and testing of knowledge bases, there are limitations so far that are tasks for future work. For the development part, only the most important SKOS properties are converted between the applications and RDF. For the quality assessment we need to develop more checks to be included in the suite. Also, there is a strong limitation in the reasoning support of SKOS-Utils. The underlying Python library *rdflib* does not support reasoning of RDF ontologies. For example, instances of SKOS concepts need to be defined explicitly (`ex:A rdf:type skos:Concept`) and not as instances of sub-classes of SKOS concepts. This also holds for other SKOS properties and classes. That way, it is required that instances under investigation need to be explicitly defined as SKOS elements. In the future, we are planning to include reasoning support for SKOS-Utils to remove this requirement.

Acknowledgments

Parts of this work were funded by German Federal Ministry for Economic Affairs and Climate Action under the ZIM program, AiF grant KK5394901GR1 (ISCO project).

References

[1] W3C, SKOS Simple Knowledge Organization System reference: http://www.w3.org/TR/skos-reference, 2009.

[2] T. Baker, S. Bechhofer, A. Isaac, A. Miles, G. Schreiber, E. Summers, Key choices in the design of simple knowledge organization system (SKOS), Journal of Web Semantics 20 (2013) 35–49.

[3] C. Mader, B. Haslhofer, A. Isaac, Finding quality issues in SKOS vocabularies, in: Proceedings of TPDL 2012, Theory and Practice of Digital Libraries, 2012. arXiv:1206.1339.

[4] O. Suominen, C. Mader, Assessing and improving the quality of skos vocabularies, Journal on Data Semantics 3 (2014). doi:10.1007/s13740-013-0026-0.

[5] J. Busse, Semantische Modelle mit Mindmaps, De Gruyter Saur, Berlin, Boston, 2014, pp. 115–127. doi:doi:10.1515/9783110312812.115.

[6] J. Lacasta, J. Nogueras-Iso, F. Lopez-Pellicer, P. Muro-Medrano, F. Zarazaga, ThManager: An open source tool for creating and visualizing SKOS, Information Technology and Libraries 26 (2007) 39–51. doi:10.6017/ital.v26i3.3274.

[7] W3C, SPARQL 1.1 recommendation: http://www.w3.org/TR/sparql11-query, 2013.

[8] C. Caracciolo, A. Stellato, A. Morshed, G. Johannsen, S. Rajbhandari, Y. Jaques, J. Keizer, The AGROVOC linked dataset., Semantic Web 4 (2013) 341–348.

A. Online Resources

SKOS-Utils is still under development and we invite collaborators to join the development team. The sources are available via GitHub: `https://github.com/denkbares/SKOS-Utils`