VoCol: An Agile Methodology and Environment for Collaborative Vocabulary Development

Niklas Petersen[1], Lavdim Halilaj[1], Christoph Lange[1], and Sören Auer[1]

University of Bonn & Fraunhofer IAIS, Germany
`niklas.petersen@iais-extern.fraunhofer.de`, `halilaj@iai.uni-bonn.de`, `math.semantic.web@gmail.com`, `auer@cs.uni-bonn.de`

Abstract Vocabularies typically reflect a consensus among experts in a certain application domain. They are thus implemented in collaboration of domain experts and knowledge engineers. Particularly the presence of domain experts with little technical background requires a low-threshold vocabulary engineering methodology. This methodology should be implementable without dependencies on complex software components, it should provide collaborators with comprehensible feedback on syntax and semantics errors in a tight loop, and it should give access to a human-readable presentation of the vocabulary. Inspired by agile software and content development methodologies, we define the VoCol methodology to address these requirements. We implemented a prototype based on a loose coupling of validation and documentation generation components on top of a standard Git repository. All of these components, even the repository engine, can be exchanged with little effort. By evaluating the usefulness of error feedback of different tools in the realistic setting of an emerging mobility vocabulary we prove, however, that *our* choice of the crucial validation component is workable.

1 Introduction

Vocabulary creation is currently a major bottleneck for the wide realisation of the Semantic Web vision. This is because vocabulary development requires a *significant* investment, which is difficult to make by a single person or organisation. If we look at current vocabularies (e.g. *LOV*[1]), we observe that they are rather simplistic. For a total of 457 vocabularies listed in LOV, a straightforward query against the LOV SPARQL endpoint tells us that the average number of classes for each vocabulary is 42 whereas the average number of properties is 59. Omitting the four vocabularies with the highest number of classes and properties, these figures decrease to 31 classes and 37 properties on average. We also observe that a large number of crucial domains is not or only superficially covered by existing vocabularies (cf. Table 1). One of the main reasons for the lack of vocabularies is also the lack of adequate methodological *and* tool support (cf. section 3). As a result, despite recent progress in vocabulary creation, for

[1] `http://lov.okfn.org`

Table 1: Identified gaps in the coverage of existing vocabularies

Domain	**Information**	**Vocabulary support missing for ...**
Spatial data	Maps	map data including road networks, traffic signs
Education	School Grades	levels of achievement, and their scales, depending on the country
Calendar	Holidays	fixed or floating holidays in each country, duration, types (e.g. non-working days, religious holidays, state holidays)
Labour market	Professions	qualifications, degrees, levels of experience; in general, and specific to countries
Nutrition	Cooking recipes	ingredients and their properties (e.g. allergenes), how to process them, possible substitutes
Do it yourself	Building/ repairing guides	required tools and materials, steps, dangerous warnings
Supply chain	Logistics, Suppliers, Products	contact information, product descriptions, production/demand forecasts

example, with the *schema.org* initiative, there is a substantial lack of comprehensive and well designed vocabularies for representing information on the Web of Data.

One possibility to tackle the vocabulary creation problem is to significantly lower the barrier for vocabulary creation and collaboration. The *VoCamp movement*[2] addresses the social perspective of lowering the vocabulary creation barrier at least when getting new vocabularies *started*. From the *technical* perspective, we argue that light-weight and agile means can be a key to providing better methodological and tool support for vocabulary creation. Agile methodologies (such as *Scrum*, *Extreme Programming*, etc.) have meanwhile gained wide acceptance in software development. Also light-weight content creation methodologies (such as the *Wiki Way* [11]) are meanwhile widely used and drive some of the largest content authoring platforms on the Web (e.g. Wikipedia). An example where such techniques are already used in practice is the DBpedia Mappings Wiki, which uses MediaWiki for authoring the multi-lingual DBpedia ontology.[3] However, the DBpedia ontology creation process is tailored specifically for DBpedia and might not be that efficiently applicable in other settings.

Few suitable solutions for developing vocabularies in collaboration of domain experts and knowledge engineers exist to date. None of them addresses requirements of vocabulary development efforts involving distributed groups of heterogeneous stakeholders (as detailed in section 2). Limitations include a high cognitive overhead for domain experts, a lack of version control or project man-

[2] `http://vocamp.org/wiki/WhatIsVoCamp`
[3] `http://mappings.dbpedia.org`

agement support, limited support for the feature set of contemporary ontology languages, or limited reusability of the ontology in automated workflows such as validation or publication as linked data. For an in-depth discussion of existing tools, see section 3.

In order to address the lack of vocabulary development support, we designed *VoCol*, a low-threshold agile methodology for collaborative vocabulary development. Inspired by agile software and content development methodologies, we define the VoCol methodology to address these requirements. We implemented a prototype based on a loose coupling of collaboration, authoring, project management, validation, documentation and visualization generation components on top of a standard Git repository. All of these components, even the repository engine, can be exchanged with little effort to cater for specific use cases. Through its continuous vocabulary component integration and verification functions, VoCol can be seen as an analogon to *continuous integration* in software engineering.

We successfully deployed and used VoCol in the *MobiVoc*[4] initiative, which aims at creating an open, standardised vocabulary[5] for *future-oriented mobility solutions*. MobiVoc is an initiative of the ITA, a trade association of information technology providers for the automotive industry. Partners of MobiVoc include automotive, IT and business consultancy companies, as well as research institutes. Developing the MobiVoc vocabulary involves members of these partner organisations in the two main roles of domain experts and knowledge engineers.

The article is structured as follows: We identify requirements, which guided the VoCol conception in section 2. We analyse and compare existing vocabulary authoring environments in section 3. The principles, roles and techniques of the methodology are outlined in section 4. We describe our implementation of integrated tool support for all aspects of the VoCol methodology in section 5. A preliminary evaluation is discussed in section 6 before we conclude with an outlook on future work in section 7

2 Requirements

The creation of VoCol was triggered by the requirements of the *MobiVoc consortium*. The aim of the consortium is to support the mobility of people through the mobility of data by developing a comprehensive vocabulary for all aspects of mobility ranging from map data, over points-of-interest to gas stations, electric charging points and traffic management. The consortium comprises a number of stakeholders (car manufacturers, researchers, IT companies, public administrations), which send representatives with different backgrounds to the working groups. The following requirements for the MobiVoc collaborative vocabulary development environment emerged from the bi-weekly phone conferences that the MobiVoc partners have held since autumn 2014.

[4] http://www.mobivoc.org/

[5] We use "vocabulary" as a synonym for "lightweight ontology" and assume that vocabularies are implemented in RDF Schema or a light profile of OWL.

Low-threshold collaboration: The environment should impose a low threshold on new collaborators, particularly on domain experts without knowledge of the formal and technical details of ontology languages.

Web frontend: The environment should be accessible with a web browser, not *requiring* the installation of any client-side tools. This contributes to keeping the threshold low for non-technical users.

Validation: In particular, the environment should tolerate errors in syntax and semantics, but it should detect them, report them in a comprehensible way and make experienced knowledge engineers aware of them.

Documentation generation: After successful validation, a human-readable documentation should be generated from the vocabulary, particularly suitable for inspection by domain experts.

Linked Open Data (LOD) publication: In parallel to the documentation, a machine-comprehensible version of the vocabulary should be published as RDF/XML linked open dataset.

Pluggable workflow: For all steps of the workflow (syntactic/semantic validation, generation of documentation or LOD, etc.) it should be possible to plug the most suitable implementation into the overall environment.

Possibility to use rich clients: Using the web frontend required above should not be the only possibility to collaborate. Experienced knowledge engineers should have the possibility to apply powerful client-side tools (e.g. offering advanced visualisation, validation, reuse from other vocabularies, or natural language technology) to the vocabulary sources without having to use an import/export mechanism and running the risk of working on an outdated version.

Version control: As the MobiVoc vocabulary is still emerging, and being designed in an agile process with a flat hierarchy open to new team members, it should be easy to see when and why design decisions were taken, and to revert old revisions.

Few technical dependencies: Setting up a running environment should require little effort.

3 Related Work

To keep our review of related work focused to the MobiVoc requirements introduced above, we only consider environments primarily dedicated to collaborative editing of vocabularies or ontologies. For example, we neither consider environments for authoring taxonomies or thesauri, nor document annotation environments with the possibility to export an ontology (such as certain semantic wikis).

WebProtégé [13] provides a collaborative web frontend for a subset of the functionality of the Protégé OWL editor[6]. The aim of WebProtégé, to lower the threshold for ontology development and to provide a collaboration environment, is similar to ours. Yet, the proposed web interface still requires a background

[6] `http://protege.stanford.edu/`

Table 2: Comparison of related systems

Requirement	WebProtégé	Neologism	DBpedia[a]	VOCREF
Syntax validation	n/a[b]	n/a[b]	−	−
Semantic validation	−	−	−	−
Usable with browser only	+	+	+	+
Combinable w/ client-side tools	+	+	−	+
Documentation generation	−	+	+	−
Linked Data publication	−	+	+	−
Version control	+	−	+	+

[a] DBpedia Mappings Wiki
[b] GUI prevents introduction of syntax errors

in knowledge modeling for any interested individual. While customization is possible in form of plugins, very few exist compared its desktop version.

Neologism [2] is a vocabulary publishing platform, with a focus on ease of use and compatibility with Linked Data principles. Neologism focuses more on vocabulary publishing and less on collaboration. For example, versioning and multiple users are not supported.

The *DBpedia Mappings Wiki*[7] uses MediaWiki for creating and curating a multilingual, multidomain ontology. The Mappings Wiki uses MediaWiki infobox templates for describing ontology classes. The ontology meanwhile covers 125 languages, 685 classes and 61,459 properties. created by dozens of collaborators.

VOCREF (Vocabulary and Ontology Characteristics Related to Evaluation of Fitness), a project of the hackathon at the 2014 Ontology Summit, is a vocabulary that was created in a collaborative process in a GitHub repository.[8] The collaborators, all of whom were technically skilled knowledge engineers, decided to use the OWL 2 Functional-style Syntax for their source files because it allows for writing exactly one ontology axiom per source line, which makes it well suited for version control. During the implementation, the collaborators committed 85 revisions and created 39 issues. These are the only observations relevant for *us*; the rest of the final report on VOCREF focuses on the vocabulary itself [14].

4 Methodology

Inspired by the portrayal of agile methodologies in [1], we developed our own version (Figure 1) for the different elements which are part of a collaborative vocabulary development environment to gain a better understanding.

The requirements collected in section 2 imply that we need to support two different tasks. First, we want to empower people with little technical expertise

[7] `http://mappings.dbpedia.org`
[8] See `http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2014_Hackathon_OntologicalCatalogue` for the project.

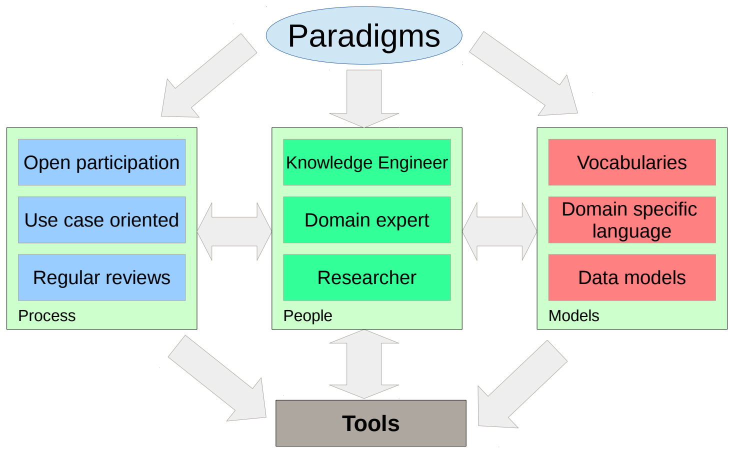


Figure 1: Elements which interact in an collaborative vocabulary environment based on [1, 8]

to contribute their domain knowledge into the vocabulary and second, to have an environment which automatically validates these contributions and either return some feedback or a new version of the vocabulary. Indeed, the former is similar to what the *wiki way* [11] content creation methodology (underlying wikis) aims to achieve and the latter goal of automatic feedback and verification is similar to what software development methodologies (such as Scrum [3]) tackle. This discovery motivated us to take these methodologies as a basis for the creation of the VoCol methodology. Therefore, we propose a methodology that borrows the following principles from the wiki way (W; cited from [12]) and agile programming (A; cited from [3]):

1. **Easy and quick editing (W)** for a low entry barrier: Contributors can edit the vocabulary by using a plain-vanilla web browser without the need of installing specific vocabulary editors.
   (a) **Openness (W)** Any interested user may edit or use the vocabulary without any restrictions. In keeping with the motto: 'make bad edits easy to correct, rather than hard to make'[9]
   (b) **Usage of a markup language (W)** Choose the simplest markup language for creating a vocabulary to facilitate non-technical users to participate in the development.
   (c) **Close, daily cooperation between business people and developers (A)** Domain experts and developers work on the same vocabulary version.
   (d) **Simplicity – the art of maximising the amount of work not done – is essential (A)** Focus on actual errors of the original source files

[9] `http://meta.wikimedia.org/wiki/The_wiki_way`

2. **Regular adaption to changing circumstances (A)** Modules of the vocabulary can be adapted flexibly. Also, components of the collaboration environment can be replaced by other components better suited to the task at hand.
   (a) **Version control (W)** Changes on the vocabulary are being tracked using a version control system.
   (b) **Welcome changing requirements, even late in development (A)** (affects both vocabulary and technical architecture)
   (c) **Projects are built around motivated individuals, who should be trusted (A)** By having a flat hierarchy, there are no restrictions on editing the vocabulary for any individual.
   (d) **Self organised teams (A)** (affects technical architecture)
3. **Customer satisfaction by rapid delivery of useful software (A)** Automatic publication of every new valid vocabulary version.
   (a) **Focus on collaborative product (W)** As in [7], we consider the development of a vocabulary as an ongoing process instead of a one-time activity.
   (b) **Working software is the principal measure of progress (A)** Every new valid vocabulary version is published automatically and can therefore be used by any individual.
   (c) **Continuous attention to technical excellence and good design (A)** After every change, the new version is being evaluated with feedback in case errors are introduced by the editor.

Figure 2 shows the generic architecture of any system that implements the VoCol methodology; algorithm 1 “zooms” into the details of the part of the process that any VoCol implementation should automate.

In section 2 we have pointed out limitations of existing visual IDEs. As we believe that some of these limitations are inherent, e.g. that the choice of underlying language determines the cognitive overhead, and as we would like to enable the use of alternative rich client tools, whose combination with import/export from an IDE leads to synchronisation issues in a collaborative setting. Therefore, VoCol does not require developing yet another visual environment but is designed to be implemented using a plain text representation of the vocabulary, which is managed by a mature version repository engine. Indeed contemporary version control systems can only reliably handle editing conflicts in plain text formats.

From the point of view of the related approaches introduced in section 3, we thus took inspiration from the VOCREF collaborative effort but emphasise domain experts as a target audience. Further reasons against using the concrete systems Neologism or WebProtégé for implementing the VoCol methodology are that Neologism is no longer actively maintained, and that WebProtégé is not yet sufficiently extensible by plugins (e.g. for vocabulary validation or publication). Its development is still in progress, and its developers advise ‘not to develop

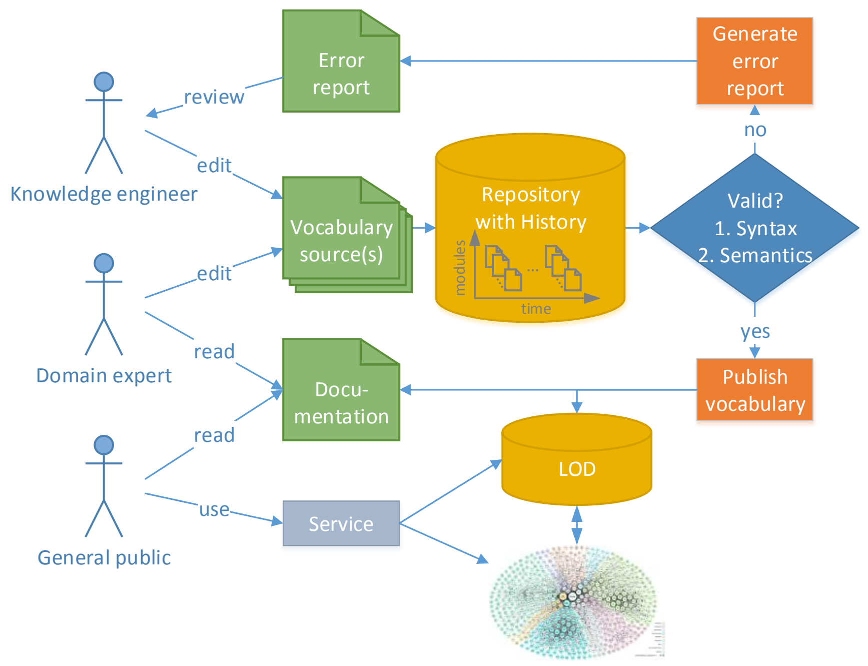


Figure 2: System architecture implementing the VoCol methodology

against it just yet'[10]. Finally, the MediaWiki-based approach of curating the DBpedia Mappings ontology is suitable only for users familiar with MediaWiki.

```
Data: Vocabulary repository
Result: Validation and possible publication of the new vocabulary ;
if new vocabulary revision exists then
    if new vocabulary revision validates then
        publish new human-friendly documentation ;
        publish new machine-comprehensible LOD ;
    else
        if errors have not been reported already then
            report errors to the revision author ;
        end
    end
end
```
Algorithm 1: Validation and possible publication of a new vocabulary

[10] `http://protegewiki.stanford.edu/index.php?title=WebProtegeDevelopersGuide&oldid=12158`

Table 3: Component choices

Components	Implementation
Vocabulary repository	Git repository
Vocabulary authoring	GitHub/GitLab UI[a], ordinary text editor
Vocabulary validation	rapper[b]
Documentation	schemaorg publication engine[c]
LOD	Apache (with static RDF/XML files)
Error report generation	GitHub/GitLab issues
Visualisation	WebVOWL

[a] `https://github.com/`, `https://about.gitlab.com/`
[b] `http://librdf.org/raptor/rapper.html`
[c] `https://github.com/schemaorg/schemaorg`

Roles. VoCol supports heterogeneous groups of stakeholders with various roles:

Domain experts have in-depth knowledge of the domain targeted by the vocabulary, but lack expertise in modeling and knowledge representation.
Knowledge engineers support the knowledge representation and modeling functions, ensure the adherence to standards and best-practices and ensure consistency of the vocabulary on a conceptual level.
Application developers are building software applications involving user interface components or data adhering to the vocabulary.
Researchers support the vocabulary development from a research perspective.

5 Implementation

For our concrete implementation of the architecture introduced above, we chose a set of state-of-the-art components and put them together using scripts and lightweight translators. In this section, we present and justify our choice of components. We build on standard technology such as the *Git* version control system for the repository, and tools from libraries such as the *Redland RDF Libraries* for validation. Table 3 provides an overview about all technologies we used. However, each component is exchangeable, and exchanging one component by another one that is better suited for a specific use case, merely requires small adaptations of the "glue", i.e. the scripts.

Vocabulary Language and Representation First of all, we have to choose a suitable vocabulary language. We primarily aim at enabling domain experts to *conceptualise* a domain, whereas a full logical formalisation, e.g. in OWL, would be left to knowledge engineers. Therefore, we consider RDF Schema suitable, as it is a lightweight language for describing vocabulary terms, including classes and properties, with little theoretical overhead.

Turtle [4], the most widely used plain text serialisation of RDF (and RDF Schema), has been designed for easy readability and writability by human users.[11] We argue that, given suitable feedback from validation tools, domain experts can be quickly trained to read and write a subset of Turtle with little effort. Our experience in MobiVoc showed that domain experts were able to read and write Turtle after just around 30 minutes of training. We recommend editing Turtle with a text editor rather than a visual IDE, because visual editors tend to change the structure of the source document on saving or exporting it. Such changes increase the risk of editing conflicts and make it harder for other collaborators to retrace the evolution of a vocabulary file over its history of revisions.

Repository Engine and Host Given the requirement to provide a low-threshold access to the repository, we recommend prioritising the availability of a web frontend over the underlying version control system. With GitLab (for self-installation) and GitHub (hosted, i.e. even less effort to set up) being widely in use[12], Git becomes a preferred repository engine for implementing the VoCol methodology.

These web frontends for Git particularly enable users who do not want or do not know how to install or how to use a Git client to contribute by editing files in the repository from the browser. Besides giving access to the files in the repository and their revision history, these frontends furthermore facilitate *project management* (by offering an issue tracker and the possibility to assign users to teams), and expose most of their functionality via a web service API so that it can be controlled programmatically.

Triggering the Validation and Publication Implementing algorithm 1, first of all, requires detecting new revisions of the repository, or, in Git terminology, responding to pushes to the central repository server and inspecting their commits. On a self-hosted Git server, one can set up *hooks*, i.e. scripts that the server calls when certain actions, such as pushes, occur.[13] In GitHub-hosted repositories, the Webhooks API[14] gives limited access to this functionality.

We chose to use GitHub and set up a Git *client*, implemented using the PyGithub API[15] and running on a server under our control, which communicates in regular time intervals with the repository server and collects, if there are any, the latest commits. It checks whether these commits affected vocabulary source files and collects the most recent version of each affected file, as well as all users who have contributed to the file in any of the commits in the time interval. The reason for this is that, if user Alice introduces an error into a file and Bob introduces another one, it is hard to implement an automated decision

[11] The specification of the N3 superset of Turtle explicitly states the design goal "to be as readable, natural, and symmetrical as possible" [5]

[12] largest code host on the planet according to `https://github.com/features/`

[13] `http://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks`

[14] `https://developer.github.com/v3/repos/hooks/`

[15] `http://jacquev6.net/PyGithub/`

procedure that would check whether or not Bob's edit fixed Alice's error. Our implementation prefers false positives over false negatives and therefore notifies all users that could in principle be concerned, relying on the collaborators to use the project management functionality of the issue tracker to sort out the responsibility for fixing errors.

Validation and Error Reporting Each file affected by a recent commit is validated. In principle, any validator with a command-line interface is suitable. Thanks to command-line HTTP clients such as *wget* or *curl*, this includes validators with a web service interface. The *syntax* of a vocabulary can, in principle, be validated by any RDF parser; for the command line there are, e.g., *rdfcat* from the Jena library, or *rapper* from the Redland RDF libraries. With Jena's *eyeball* there is also a command-line *semantics* validator, which can, e.g., check for "likely cardinality violations" and "individuals having consistent types".[16] As vocabularies are just a special case of linked datasets, one can finally employ *data quality* assessment tools, such as our own Luzzu [9], which has a web service interface. Quality metrics implemented in Luzzu that are relevant to vocabularies include a check for the existence of human-readable labels or comments. The wide availability of command-line validators and also converters for different RDF serialisations opens up the VoCol methodology for other vocabulary languages. If, e.g., the additional expressivity of OWL is desired, the OWLTools[17] command line interface to the OWL API, provides validation and supports human-friendly serialisations such as the OWL Manchester Syntax [10].

The main prerequisite for employing a certain validator in a VoCol implementation is that precise information about errors can be extracted from its output. Such information typically includes filename, line number and the type of error. For validation error, our repository client first checks whether a report exists already, by comparing these fields. If not, it generates a new report.

Our implementation creates GitHub issues whose title contains the error message. The body of the issue includes a link to the file in the version of the most recent commit (highlighting the affected line), links to earlier commits in the current validation round that affected the same file, and it names the last contributors who edited the file, plus the members of the "knowledge engineers" team. GitHub supports collaboration by not only supporting one user to be the assignee of an issue, but also sending notifications to all users addressed with the `@user` syntax.

Publication for Humans and Machines After successful validation, we publish the vocabulary in human and machine friendly ways.

We first merge all Turtle source modules into a single file. From this file, we generate an HTML+RDFa representation as required, using the schemaorg documentation generator. The schemaorg documentation generator serves the

[16] `https://jena.apache.org/documentation/tools/eyeball-manual.html`
[17] `https://owltools.googlecode.com/`

human-readable HTML documentation generated from this input from a local web server, from which we download all pages into static HTML files. We also generate a single-file RDF/XML version of the vocabulary. We publish the resulting HTML and RDF/XML files using a web server configuration that makes them available under the URIs of the respective vocabulary terms and performs content negotiation between HTML and RDF/XML according to the best practices for publishing vocabularies [6].

For visualization purposes we use WebVOWL, a web application tool for the user-oriented visualization of ontologies which implements the Visual Notation for OWL Ontologies (VOWL) by providing graphical depictions for elements of the OWL Web Ontology Language [18].

Deployment We deployed our software as a VirtualBox virtual machine image, which can be installed with little effort. Using Vagrant, a tool for building and managing virtualised development environments[19], we make the process of setting up this image reproducable and document it at the same time. All implementation is available from the VoCol repository at `https://github.com/mobivoc/vocol/`.

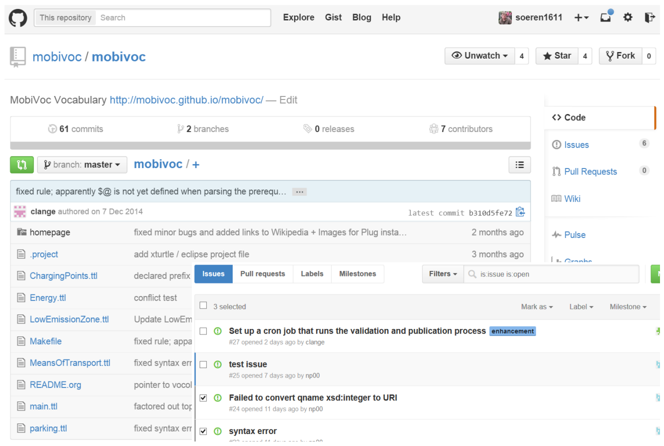
6 Evaluation

Evaluating methodologies and tools for knowledge engineering is inherently difficult. A thorough evaluation can only be performed by using the methodology and collaboration environment in a variety of different projects over a long period of time and assessing usage results, performing user interviews as well as assessing the quality of the collaboration and resulting vocabularies. The VoCol methodology and collaboration environment was developed as a result of the requirements of the MobiVoc consortium. It was meanwhile used by this consortium over a period of several months, but without sufficient duration and intensity for supporting a comprehensive, empirical study. Nevertheless and in the spirit of a conference publication sharing timely research results, we wanted to already share the VoCol idea with the community.

In the following we first present a *methodology* for choosing the right components to integrate in an implementation of the VoCol methodology, and then share the results of applying this evaluation methodology to the central component that implements vocabulary validation.

To make sure that a component of a VoCol implementation is fit for its purpose, it has to be evaluated against the key principles of the VoCol methodology (cf. section 2). The overall suitability of a component for collaborating on a vocabulary can only reliably be assessed on a sufficiently large *revision history*, rather than just on the most recent version. In practice this requires checking out from the repository all relevant revisions (i.e. those that affected the vocabulary

[18] `http://vowl.visualdataweb.org/webvowl.html`
[19] `https://www.vagrantup.com/`



mobivoc / mobivoc

MobiVoc Vocabulary http://mobivoc.github.io/mobivoc/ — Edit

61 commits · 2 branches · 0 releases · 7 contributors

branch: master · mobivoc / +

fixed rule; apparently $@ is not yet defined when parsing the prerequ… …

clange authored on 7 Dec 2014 · latest commit b310d5fe72

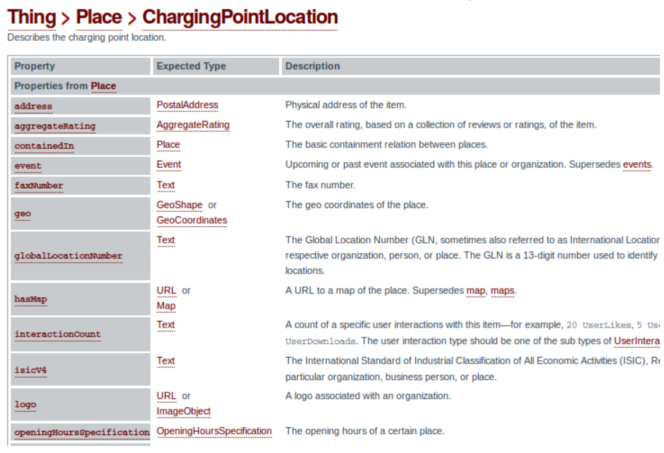
homepage	fixed minor bugs and added links to Wikipedia + Images for Plug insta…	2 months ago
.project	add xturtle / eclipse project file	3 months ago
ChargingPoints.ttl	declared prefix	
Energy.ttl	conflict test	
LowEmissionZone.ttl	Update LowEm	
Makefile	fixed rule; appa	
MeansOfTransport.ttl	fixed syntax err	
README.org	pointer to vocol	
main.ttl	factored out top	
parking.ttl	fixed syntax err	

Issues · Pull requests · Labels · Milestones · Filters · is:issue is:open

3 selected · Mark as · Label · Milestone

- [ ] Set up a cron job that runs the validation and publication process `enhancement`
  #27 opened 2 days ago by clange
- [ ] test issue
  #25 opened 7 days ago by np00
- [x] Failed to convert qname xsd:integer to URI
  #24 opened 11 days ago by np00
- [x] syntax error

(a) VoCol collaboration environment on GitHub (source and issue views).

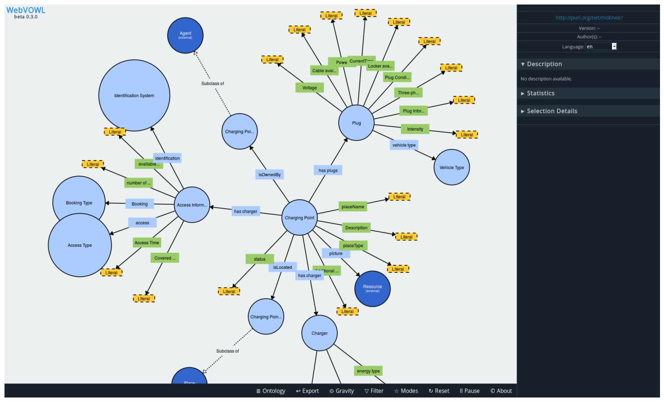


Thing > Place > ChargingPointLocation

Describes the charging point location.

Property	Expected Type	Description
Properties from Place		
address	PostalAddress	Physical address of the item.
aggregateRating	AggregateRating	The overall rating, based on a collection of reviews or ratings, of the item.
containedIn	Place	The basic containment relation between places.
event	Event	Upcoming or past event associated with this place or organization. Supersedes events.
faxNumber	Text	The fax number.
geo	GeoShape or GeoCoordinates	The geo coordinates of the place.
globalLocationNumber	Text	The Global Location Number (GLN, sometimes also referred to as International Locatior respective organization, person, or place. The GLN is a 13-digit number used to identify locations.
hasMap	URL or Map	A URL to a map of the place. Supersedes map, maps.
interactionCount	Text	A count of a specific user interactions with this item—for example, 20 UserLikes, 5 Us UserDownloads. The user interaction type should be one of the sub types of UserIntera
isicV4	Text	The International Standard of Industrial Classification of All Economic Activities (ISIC), R particular organization, business person, or place.
logo	URL or ImageObject	A logo associated with an organization.
openingHoursSpecification	OpeningHoursSpecification	The opening hours of a certain place.

(b) Schema.org style documentation of classes and properties.



(c) Vocabulary Visualisation using WebVOWL.

Figure 3: Interfaces to the VoCol environment

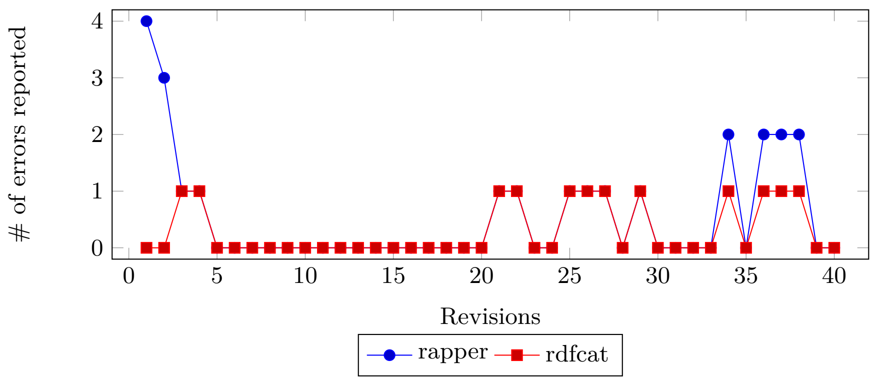


Figure 4: Number of errors reported by rapper and rdfcat

source), applying the component in question to these sources, and assessing the usefulness of the component's output.

In the concrete case of a syntax validator the central quality criterion is the precision and comprehensibility of its error messages. Also, as we are not validating errors inside an IDE, but only when a user pushes possibly several commits at once, we are interested in detecting more than one syntax error at a time, i.e. we are interested in a parser that tolerates minor errors and continues processing the input. Concretely, we aimed at measuring whether *rdfcat* or *rapper* is better suited for this job by determining the number of errors they reported in a single run. Figure 4 shows the result of this comparison, clearly indicating that rapper is better suited for use in a VoCol implementation. Note that there is still further space for improvement, as in many revisions the source files actually contained a much larger number of errors.

7 Conclusions & Future Work

In this paper we presented our approach to facilitate the collaborative development of vocabularies by combining and integrating a number technologies. We devised the VoCol methodology, presented a tool architecture supporting the methodology, showcased an implementation integrating various components and demonstrated the viability of the approach with the vocabulary developed by the *MobiVoc* initiative. We hope that with VoCol we can contribute to solving one of the most pressing obstacles for a wider, industrial use of semantic technologies. In particular, the agile and collaborative nature of VoCol, which facilitates the engagement of various stakeholders from different organizations and with different skill levels has the potential to dramatically improve the efficiency and effectiveness of vocabulary development initiatives.

Based on the experiences with the MobiVoc vocabulary development initiative, we plan to start vocabulary development activities in other domains such as for example manufacturing supply chains. We plan to automatize and simplify further steps of the vocabulary development. For example, the integration and use of visual editors is an interesting direction of further work. Once VoCol is used in a variety of vocabulary development initiatives by a large number of users, we aim to perform a comprehensive empirical study regarding the efficiency end effectiveness of vocabulary development in the light of various factors such as vocabulary/community size.

References

1. Auer, S. RapidOWL – A Methodology for Enabling Social Semantic Collaboration. In: *Semantic Web Engineering in the Knowledge Society*. Ed. by J. Cardoso, M. D. Lytras. Idea Group, 2009.
2. Basca, C. et al. Neologism: Easy Vocabulary Publishing. In: *Scripting and Development for the Semantic Web (SFSW)*. CEUR-WS 368. Aachen, 2008.
3. Beck, K. et al. Manifesto for agile software development. 2001. `http://agilemanifesto.org/` (visited on 2015-01-15).
4. Beckett, D. et al. RDF 1.1 Turtle. Terse RDF Triple Language. Recommendation. W3C, 2014. `http://www.w3.org/TR/turtle/`.
5. Berners-Lee, T., Connolly, D. Notation3 (N3): A readable RDF syntax. Team Submission. W3C, 2011. `http://www.w3.org/TeamSubmission/n3/`.
6. Berrueta, D., Phipps, J. Best Practice Recipes for Publishing RDF Vocabularies. Tech. rep. W3C, 2008. `http://www.w3.org/TR/2008/NOTE-swbp-vocab-pub-20080828/`.
7. Braun, S. et al. Ontology Maturing: a Collaborative Web 2.0 Approach to Ontology Engineering. In: *CKC Workshop at WWW2007*. CEUR-WS 273. 2007.
8. Cockburn, A. Selecting a project's methodology. In: IEEE software 17(4) (2000).
9. Debattista, J. et al. LUZZU – A Framework for Linked Data Quality Assessment. arXiv: `1412.3750 [cs.DB]`.
10. Horridge, M., Patel-Schneider, P. F. OWL 2 Web Ontology Language: Manchester Syntax. Tech. rep. W3C, 2009.
11. Leuf, B., Cunningham, W. The Wiki Way: Collaboration and Sharing on the Internet. Addison-Wesley, 2001.
12. Moskaliuk, J. Konstruktion und Kommunikation von Wissen mit Wikis. German. In: Theorie und Praxis (2008).
13. Tudorache, T. et al. WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web. In: Semantic Web 4(1) (2013).
14. Vizedom, A. Ontology-Vocabulary Characteristics Relevant to Suitability for Semantic Web & Big Data applications. In: Ontology Summit Symposium. 2014. `http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2014_Symposium#nid4CUL`.