VoCol: An Integrated Environment to Support Version-Controlled Vocabulary Development

Lavdim Halilaj[1,2(✉)], Niklas Petersen[1,2], Irlán Grangel-González[1,2], Christoph Lange[1,2], Sören Auer[1,2], Gökhan Coskun[3], and Steffen Lohmann[2]

[1] Enterprise Information Systems (EIS), University of Bonn, Bonn, Germany
{halilaj,petersen,grangel,langec,auer}@cs.uni-bonn.de
[2] Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), Sankt Augustin, Germany
{Lavdim.Halilaj,Niklas.Petersen,Irlan.Grangel-Gonzalez,Christoph.Lange, Soren.Auer,Steffen.Lohmann}@iais.fraunhofer.de
[3] Bayer Business Services, Berlin, Germany
goekhan.coskun@bayer.com

Abstract. Vocabularies are increasingly being developed on platforms for hosting version-controlled repositories, such as GitHub. However, these platforms lack important features that have proven useful in vocabulary development. We present VoCol, an integrated environment that supports the development of vocabularies using Version Control Systems. VoCol is based on a fundamental model of vocabulary development, consisting of the three core activities modeling, population, and testing. We implemented VoCol using a loose coupling of validation, querying, analytics, visualization, and documentation generation components on top of a standard Git repository. All components, including the version-controlled repository, can be configured and replaced with little effort to cater for various use cases. We demonstrate the applicability of VoCol with a real-world example and report on a user study that confirms its usability and usefulness.

Keywords: Vocabulary development · Version control system · Ontology engineering · Integrated development environment · IDE · Git · GitHub · Webhook

1 Introduction

Vocabulary development is currently a major bottleneck for the wide realization of the Semantic Web vision. It requires a significant investment, which is difficult to make by a single person or organization. Identifying the terms and concepts by finding a consensus among the involved stakeholders and defining a shared vocabulary[1] is an effective approach to tackle this problem. However, this

[1] In this work, the term "vocabulary" is used to refer to lightweight ontologies, as they are developed in initiatives like schema.org and defined by the W3C [23].

process, which we refer to as *distributed vocabulary development*, can be quite complex. In fact, the main challenge for vocabulary engineers is to work collaboratively on a shared objective in a harmonic and efficient way, while avoiding misunderstandings, uncertainty, and ambiguity.

On the other hand, *Version Control Systems* (VCS), such as *Subversion* (SVN) or *Git*, are becoming increasingly popular for vocabulary development. In our previous work, we proposed *Git4Voc* [6], a set of best practices which transfer concepts of VCSs to vocabulary development, on the example of Git. We discovered that several aspects of vocabulary development—in particular with regard to revision management, access control, and some governance issues—are already well covered by Git-based version control, especially if developers follow the proposed best practices.

Many of the current vocabulary development activities take place on repository hosting platforms like *GitHub*, *GitLab*, and *BitBucket*. In addition to mere version-controlled (e.g. Git) repositories, these platforms provide features such as change tracking (e.g. diffs), comments, issue tracking, wikis, and notifications. Examples of popular vocabulary projects that are publicly maintained on GitHub include Schema.org, *FOAF*, *BIBO*, *DOAP*, and the *Music Ontology*.[2] However, despite all benefits of developing vocabularies on repository hosting platforms like GitHub, these platforms lack important features that have proven useful in vocabulary development. In particular, they do not provide an integrated environment typically found in systems dedicated to distributed vocabulary development, such as *WebProtégé* [22] or *VocBench* [21].

We designed *VoCol* as a holistic approach for realizing a full-featured vocabulary development environment centered around version control systems. VoCol supports a fundamental round-trip model of vocabulary development, consisting of the three core activities *modeling*, *population*, and *testing*. In the spirit of test-driven software engineering, VoCol allows to formulate queries which represent competency questions for testing the expressivity and applicability of a vocabulary *a priori*. For *a posteriori* testing, it supports the automatic detection of "bad smells" in the vocabulary design by employing SPARQL patterns. For modeling, VoCol integrates a number of techniques facilitating the conceptual work, such as automatically generated documentations and visualizations providing different views on the vocabulary as well as an evolution timeline supporting traceability. For population, VoCol supports the integration of mappings between data sources (e.g., R2RML mappings to relational databases) and the vocabulary. The governance of distributed vocabulary development is supported by the access control as well as the branching and merging mechanisms of the underlying VCS.

As a result, VoCol bridges between the *conceptual* development of vocabularies and the *operational* execution in a concrete IT landscape. The implementation of VoCol is based on a loose coupling, leveraging the webhook method provided by many VCSs with tools and techniques focusing on particular aspects of

[2] See https://github.com/ + schemaorg/schemaorg, foaf/foaf, structureddynamics/Bibliographic-Ontology-BIBO, edumbill/doap, motools/musicontology, among others.

vocabulary development. By proving Vagrant and Docker containers bundling all tools and encapsulating dependencies, VoCol is easily deployable or even usable as-a-service in conjunction with arbitrary VCS installations.

The remainder of this paper is structured as follows: Section 2 introduces the fundamental round-trip model that VoCol is based on and lists requirements that are critical for distributed vocabulary development. Based on the model and requirements, we developed the VoCol system architecture that is presented in Sect. 3. Section 4 introduces an implementation of VoCol that we realized on top of Git. Section 5 reports on a qualitative evaluation of the usefulness and usability of the VoCol environment. Finally, VoCol is compared to related environments for distributed vocabulary development in Sect. 6, before the paper is concluded in Sect. 7.

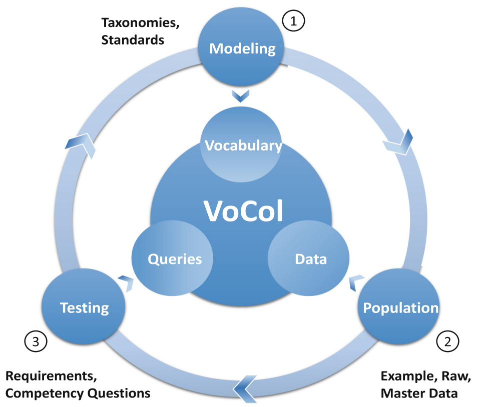
2 Round-Trip Model and Requirements

Deriving requirements for the envisioned development environment demands the clarification of our understanding of the most fundamental vocabulary development activities. A vocabulary comprises a terminology which is known as *TBox*. The creation of this terminology is realized using a logical formalism during the modeling activities [5]. This comprises the analysis and conceptualization of the domain and the specification of the vocabulary terms, such as classes, properties, and the relationships between them. Once the vocabulary modeling has been completed, the next activity is typically population. It includes the addition of actual data in line with the defined classes and properties, also known as *ABox* [3]. To verify whether the created vocabulary correctly represents the domain, a list of queries can be compiled from *competency questions* [17] and used for testing purposes. Vocabulary engineers may iterate in an incremental fashion between the modeling, population, and querying activities. In fact, these three core activities lead to the conception of *round-trip* development as illustrated in Fig. 1a.

In order to develop an integrated environment that supports the described round-trip development of vocabularies, corresponding requirements have to be identified and addressed accordingly. In our previous work on Git4Voc [6], we identified eleven requirements that are crucial for the successful adaptation of Git to vocabulary development. We gathered these requirements by aggregating insights from the state of the art and our own experiences with developing the vocabularies *MobiVoc* and *SCORVoc* on GitHub.[3] For the design of VoCol, we revised these requirements and grouped them into four categories that need to be addressed by an integrated environment that aims to support full-featured vocabulary development (cf. Fig. 1b). In the following, we briefly summarize the categories and requirements. For a more detailed description, please refer to the Git4Voc paper [6] and referenced works (i.e. [4,8,11,13,14,17,20]).

Collaboration Support: The first category contains requirements that ease collaboration in distributed settings. *R1 Governance:* Stakeholders with

[3] See https://github.com/vocol/mobivoc and https://github.com/vocol/scor.



(a) Round-trip model

1. Collaboration Support:
R1 Governance (roles, permissions, etc.)
R2 Communication (issue tracking, notifications, etc.)
R3 Provenance (revision history, semantic diffs, etc.)

2. Quality Assurance:
R4a Syntactic Validation (RDF/OWL compliance, etc.)
R4b Semantic Validation (consistency checking, etc.)
R5 Testing (competency questions, etc.)

3. User Experience:
R6 Documentation (e.g., generated HTML)
R7 Visualization (e.g., node-link diagram)
R8 Editor agnostic (serializations, normalizations, etc.)

4. Vocabulary Deployment:
R9 Machine accessibility (content negotiation, etc.)
R10 Internationalization (multilinguality, etc.)
R11 Querying (e.g., SPARQL endpoint)

(b) Categories and requirements

Fig. 1. (a) Round-trip vocabulary development supported by VoCol; (b) categories and requirements to be addressed by an integrated vocabulary development environment.

different backgrounds and levels of expertise are involved in vocabulary development. Consequently, the definition of roles and permissions is an important requirement [14,20]. *R2 Communication:* The collaborative development of vocabularies is about finding consensus among the different stakeholders. It is essential that they share ideas, make agreements, and discuss issues during the entire development life cycle [13,14]. *R3 Provenance:* It is also crucial to track changes made by the contributors [14]. Each change in the vocabulary reflects the understanding of the domain by the respective stakeholder. In case of disagreements, it is necessary to know which change has been made by whom at which time and for what reason. Furthermore, the development of vocabularies should respond to the evolution of the knowledge domain [20]. Hence, support for detecting and documenting provenance of information and semantic differences between versions is needed during the entire development process.

Quality Assurance: This category comprises requirements for the systematic checking of quality criteria that should be fulfilled by the vocabulary. *R4 Syntax, Semantic, and Constraint Validation:* Syntactic and semantic correctness as well as the application of best practices on designing vocabularies are relevant quality aspects. Providing tool support for these aspects is essential to help contributors in making fewer errors and ultimately increasing the quality of the vocabulary. *R5 Testing:* Competency Questions, i.e., questions the vocabulary must be able to answer, can be translated into queries and used as test cases for the vocabulary [17]. An integrated vocabulary development environment should provide means that allow users to execute such queries efficiently.

User Experience: This category groups requirements for enabling contributors to achieve their objectives effectively and in a user-centered manner. *R6 Documentation:* Domain experts are often team members with little technical expertise in knowledge representation and engineering tools. Thus, presenting the current state of the vocabulary in a human-friendly way is vital. *R7 Visualization:* Visualization is known to have a positive impact on the modeling, exploration, verification, and sense-making of vocabularies [11]. It is particularly helpful for domain experts, but can also provide useful insights for knowledge engineers. *R8 Editor agnostic:* In contrast to software code, vocabularies are conceptual artifacts that can be serialized in different ways. Since contributors can use various editors, which style the syntax differently, support for collaborative vocabulary development should be editor-agnostic and syntax-independent.

Vocabulary Deployment: Finally, there are requirements concerning the deployment of the developed vocabulary that also need to be taken into account by an integrated environment. *R9 Machine accessibility:* An important requirement towards realizing the vision of the web as a global information space is to provide details about the vocabulary terms in a representation that meets the requested type and format [8], thus enabling machines to process the vocabulary correctly. *R10 Internationalization:* The internationalization and localization of vocabularies should also be supported by the environment. The translation of terms into other languages enables a vocabulary to be applicable in different cultures and communities [4]. *R11 Querying:* In order to check whether the developed vocabulary is suitable for a certain use case and to easily retrieve information for a specific task, the environment should support the execution of user-defined queries.

3 System Architecture

In order to implement VoCol as an integrated environment, we developed the system architecture illustrated in Fig. 2. It follows the principles of *Component Based Software Development* (CBSD) [9], which promotes the reuse of components to develop large-scale systems. In other words, it advocates selecting the appropriate *off-the-shelf* components and assembling them into a well-defined software architecture. Following this idea, we composed VoCol from a set of smaller components according to the functionalities they provide. Each of these components is exchangeable and can be replaced by alternatives. In the following, the components are described in detail.

Version Control System: A VCS component is required for the *management of vocabulary changes*. By capturing and storing the changes, various revisions of the vocabulary are created. Contributors should work collaboratively, at best without the need of sharing a common network or the necessity of being always online. In addition, conflicts inevitably arise in environments where multiple contributors are working simultaneously and changing vocabulary terms. The VCS ensures conflict resolution and allows the integration of conflicting changes in an effective and easy way.

Since the VCS is the first component that is aware of changes, we declared it to be the core component of the overall VoCol system. Each additional component that is necessary to support vocabulary development is triggered by the

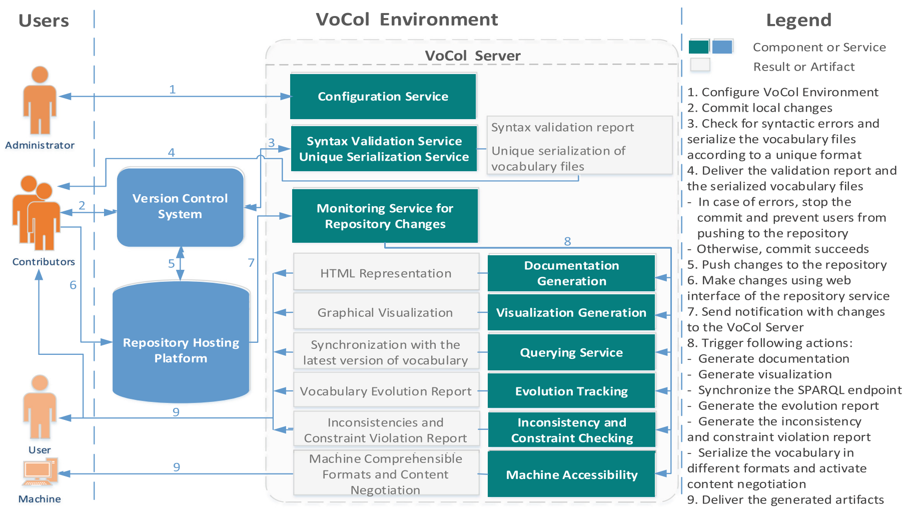

Users
VoCol Environment
Legend
VoCol Server
Component or Service
Result or Artifact
Administrator
Contributors
User
Machine
Configuration Service
Syntax Validation Service
Unique Serialization Service
Syntax validation report
Unique serialization of vocabulary files
Version Control System
Monitoring Service for Repository Changes
Repository Hosting Platform
HTML Representation
Documentation Generation
Graphical Visualization
Visualization Generation
Synchronization with the latest version of vocabulary
Querying Service
Vocabulary Evolution Report
Evolution Tracking
Inconsistencies and Constraint Violation Report
Inconsistency and Constraint Checking
Machine Comprehensible Formats and Content Negotiation
Machine Accessibility
1. Configure VoCol Environment
2. Commit local changes
3. Check for syntactic errors and serialize the vocabulary files according to a unique format
4. Deliver the validation report and the serialized vocabulary files
- In case of errors, stop the commit and prevent users from pushing to the repository
- Otherwise, commit succeeds
5. Push changes to the repository
6. Make changes using web interface of the repository service
7. Send notification with changes to the VoCol Server
8. Trigger following actions:
- Generate documentation
- Generate visualization
- Synchronize the SPARQL endpoint
- Generate the evolution report
- Generate the inconsistency and constraint violation report
- Serialize the vocabulary in different formats and activate content negotiation
9. Deliver the generated artifacts

Fig. 2. VoCol architecture and workflow

VCS. We also integrated a repository hosting platform into the VoCol environment (cf. Fig. 2), as it provides low-threshold access to the repository. It acts as the repository storage where the vocabulary files are saved and accessed. Its feature for *Access Control* authenticates users and outputs a permit or a deny message according to the set permissions. Furthermore, using the *Issue Tracker* of the repository hosting platform, contributors are able to discuss the vocabulary by proposing new terms or alternatives for existing ones. In cases where sensitive information should be transmitted, the repository hosting platform can deliver *email notifications* to private user accounts.

Syntax Validation: To ensure that the latest revision of the vocabulary in the VCS is always syntactically correct, VoCol integrates a syntax validation component. In principle, syntax validation could be executed at different stages of the overall workflow. However, with the aim to keep the requirements on the client side at a minimum level, we integrated the syntax validation as a service in the backend. It rejects syntactically incorrect commits and provides a detailed error report in those cases.

Unique Serialization Service: In a distributed environment, contributors use different editors during the development process which may produce different structures of vocabulary files. To avoid this problem, a service integrated into VoCol creates a unique serialization of vocabulary terms before the changes are pushed to the remote repository [7]. Thus, the VCS is prevented from indicating *false-positive* conflicts.

Documentation Generation: A documentation generation service creates an HTML representation of the vocabulary. This permits contributors to easily navigate through the vocabulary by providing a human-friendly overview of it.

Visualization Generation: The integrated visualization component depicts the vocabulary terms and their connections in a graphical way, and allows for the interaction with the visualization. It complements the generated documentation by particularly representing the structure, distribution, and relationships within the vocabulary.

Evolution Tracking: The VCS takes care of maintaining the revision history of the files. To detect *semantic* differences between vocabulary versions, an *evolution tracking* service is integrated into VoCol. It shows which classes and properties have been added, removed, or modified, enabling users to see the vocabulary evolution over time.

Querying Service: VoCol integrates a SPARQL endpoint synchronized with the latest version of the vocabulary. During testing, queries derived from competency questions [17] can be used to verify whether the vocabulary fulfills the domain requirements. These queries are stored in the repository and are preloaded in the query user interface.

Inconsistency and Constraint Checking: After the changes have been pushed to the remote repository, validations of semantic inconsistencies and constraint violation are performed. As a result, two reports with detailed information on respective findings are generated and can be used for corrections.

Machine Accessibility: Using *content negotiation* and dereferenceable URIs, VoCol delivers various machine-comprehensible representations. By specifying the content type in the HTTP header along with the resource URI, the vocabulary can be accessed by different software agents compliant with Linked Data principles.

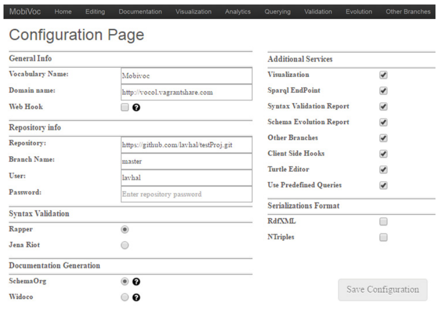
Monitoring Service: Repository hosting platforms typically expose most of their functionality via web service APIs, so that it can be controlled programmatically. Any change to the repository is delivered as a payload event to a monitoring service listening on VoCol. As a consequence, the services for documentation generation, visualization, evolution tracking, querying, etc. are automatically invoked.

Configuration Service: This service provides a *graphical user interface* to facilitate the configuration of VoCol. The system administrator can choose from various tools for syntax validation and documentation generation. Furthermore, the other services can be activated or deactivated simply by selecting the corresponding checkboxes.

4 Implementation

We use the VCS *Git* at the core of the implementation, together with a set of integrated components providing functionalities for syntax validation, visualization, documentation and evolution report generation, querying, etc.[4]

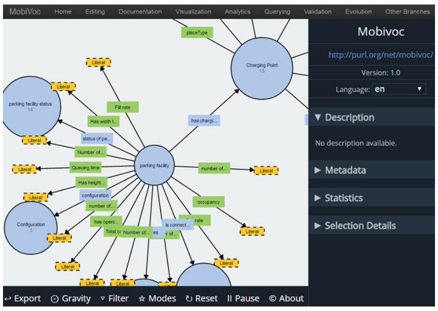
[4] A live demo of VoCol is available at http://vocol.visualdataweb.org.



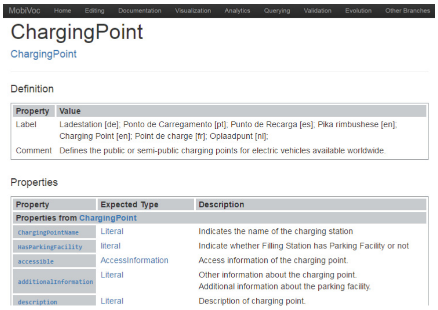
Configuration Page

General Info	
Vocabulary Name:	Mobivoc
Domain name:	http://vocol.vagrantshare.com
Web Hook	☐
Repository info	
Repository:	https://github.com/lavhal/testProj.git
Branch Name:	master
User:	lavhal
Password:	Enter repository password
Syntax Validation	
Rapper	●
Jena Riot	○
Documentation Generation	
SchemaOrg	●
Widoco	○
Additional Services	
Visualization	✓
Sparql EndPoint	✓
Syntax Validation Report	✓
Schema Evolution Report	✓
Other Branches	✓
Client Side Hooks	✓
Turtle Editor	✓
Use Predefined Queries	✓
Serializations Format	
RdfXML	☐
NTriples	☐

(a) Configuration



(b) Visualization



ChargingPoint

ChargingPoint

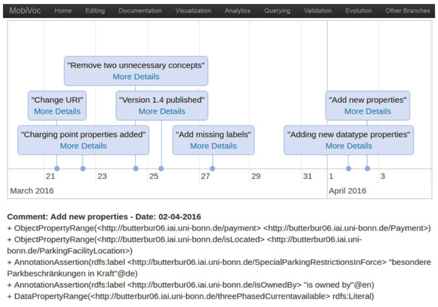
Definition

Property	Value
Label	Ladestation [de]; Ponto de Carregamento [pt]; Punto de Recarga [es]; Pika rimbushese [en]; Charging Point [en]; Point de charge [fr]; Oplaadpunt [nl];
Comment	Defines the public or semi-public charging points for electric vehicles available worldwide.

Properties

Property	Expected Type	Description
Properties from ChargingPoint		
`ChargingPointName`	Literal	Indicates the name of the charging station
`HasParkingFacility`	literal	Indicate whether Filling Station has Parking Facility or not
`accessible`	AccessInformation	Access information of the charging point.
`additionalInformation`	Literal	Other information about the charging point. Additional information about the parking facility.
`description`	Literal	Description of charging point.

(c) Documentation



Comment: Add new properties - Date: 02-04-2016
+ ObjectPropertyRange(<http://butterbur06.iai.uni-bonn.de/payment> <http://butterbur06.iai.uni-bonn.de/Payment>)
+ ObjectPropertyRange(<http://butterbur06.iai.uni-bonn.de/isLocated> <http://butterbur06.iai.uni-bonn.de/ParkingFacilityLocation>)
+ AnnotationAssertion(rdfs:label <http://butterbur06.iai.uni-bonn.de/SpecialParkingRestrictionsInForce> "besondere Parkbeschränkungen in Kraft"@de)
+ AnnotationAssertion(rdfs:label <http://butterbur06.iai.uni-bonn.de/isOwnedBy> "is owned by"@en)
+ DataPropertyRange(<http://butterbur06.iai.uni-bonn.de/threePhasedCurrentavailable> rdfs:Literal)

(d) Evolution

Fig. 3. Screenshots of selected VoCol services

4.1 Configuration

We developed a service that allows the utilization of VoCol for different application scenarios. Using this service, the system administrator configures VoCol by entering the details of the vocabulary repository (i.e., repository URL, user credentials, etc.) in the graphical user interface (cf. Fig. 3a). Next, different tools can be chosen for syntax validation and documentation generation. Via check-boxes, services for visualization, evolution report generation, querying, etc. can be selected for automatic execution from VoCol. The administrator defines the main branch of the repository by entering the value in the *Branch Name* field. For this branch, all selected services will be provided by VoCol. If the option *Monitor Other Branches* is chosen, some of the services are performed on the other branches of the repository too.

Furthermore, the option *Turtle Editor* can be selected to integrate a tool for the online editing of Turtle files into the vocabulary repository [16]. The option *Predefined Queries* indicates that queries defined in files with the extension *.rq* will automatically be loaded into the SPARQL interface. Finally, all serialization formats that VoCol should support via content negotiation can be selected.

VoCol detects the used repository hosting platform (GitHub, BitBucket, etc.) based on the URL entered for the vocabulary repository, and accesses the platform's API to create a *webhook*. This hook contains the address of the VoCol server to which the repository hosting platform will henceforth send information about any push event.

4.2 Client-Side Tasks

Client-side tasks refer to the tasks that are performed before pushing to the repository. To reduce the efforts needed for subsequent corrections, VoCol validates the syntax before pushing the changed files to the repository. An adapted *pre-commit* hook posts vocabulary files that have been changed with tools like *Protégé* or *TopBraid Composer*[5] from the local user repository to the VoCol server. First, the server validates the vocabulary files for syntactic errors. If the validation fails, the user receives a detailed error description, including the file name, the affected lines in the files, and the type of error. If the syntax validation succeeds, a unique serialization of the vocabulary files is created using the SerVCS service [7] we developed on top of the RDF serialization tool Rdf-toolkit[6]. As a result, the vocabulary elements will be serialized in an alphabetic order, which reduces the number of false-positive conflicts indicated by the VCS during the merging process. Additionally, the integrated *TurtleEditor* [16] can be used to edit the vocabulary files directly on the repository hosting platform. Following the idea of a *just-in-time debugger*, this editor implements an instant validator that immediately reports on all found syntax errors. Furthermore, it provides auto-completion of vocabulary terms according to the declared namespaces.

4.3 Server-Side Tasks

Server-side tasks refer to tasks related to the validation and publication of artifacts in human and machine-comprehensible formats that are performed after a Git push event.

Triggering Changes on the Repository: Using the *PubSubHubbub* protocol[7], on each push event, the repository hosting platform delivers a payload with information about the last commit to a server subscribed to it. The *Monitoring Service* implemented in VoCol receives the payload and pulls the vocabulary from the remote repository.

Validation and Error Reporting: Next, the *Syntax Validation* service validates each file for syntax errors using tools like *Rapper* or *Jena Riot*[8]. This task is rerun on the server side to avoid further processing of vocabularies with syntax errors, which can happen if users do not validate the syntax on their commit. If the validation fails, an HTML document is created with detailed information about the errors.

[5] http://protege.stanford.edu, http://www.topquadrant.com/composer/.
[6] https://github.com/edmcouncil/rdf-toolkit.
[7] https://pubsubhubbub.appspot.com.
[8] http://librdf.org/raptor/, https://jena.apache.org/documentation/io/.

Publishing the Artifacts for Humans and Machines: If the syntax validation process is passed successfully, all vocabulary files are merged into a single file. After that, the following tasks are performed automatically; they generate updated artifacts for the evolution report, documentation, and visualization.

Documentation Generation: A human-friendly documentation of the vocabulary is generated using tools such as the documentation generator of Schema.org or *Widoco*[9].

1. Using Schema.org: We developed an HTML generator that creates an RDFa representation for each element of the vocabulary. Next, the content is rendered by the Schema.org tool as one page per resource, which makes the elements dereferenceable. An example of an HTML page generated for a vocabulary term (*ChargingPoint*) is shown in Fig. 3c.
2. Using *Widoco:* A single HTML page listing all elements of the vocabulary is generated by Widoco. This provides the user with a complete overview of the vocabulary that can be easily navigated and searched.

Visualization Generation: The vocabulary is visualized using the web application *WebVOWL* [10]. WebVOWL implements the Visual Notation for OWL Ontologies (VOWL) by graphically representing the vocabulary terms and their relations in a dynamic node-link diagram. An excerpt of a generated visualization is shown in Fig. 3b.

Evolution Tracking: When semantic differences between versions of the vocabulary exist, an evolution report is generated using the tool Owl2vcs [25]. It uses algorithms for structural diffs and three-way merge tools along with OWL 2 direct semantics. The application of direct semantics eliminates problems with blank nodes and allows comparing ontologies axiom by axiom. The report contains each point in time when a new vocabulary revision has been pushed, and lists semantic changes like the *addition*, *removal*, or *modification* of elements, as shown in Fig. 3d.

Machine Accessibility: Machine-comprehensible formats of the vocabulary, such as *Turtle* and *RDF/XML* produced by Rapper, are delivered through a web server configured to perform *content negotiation* according to the best practices for publishing vocabularies[10]. As a result, machines are provided with the latest version of the vocabulary at any time.

Querying Service: An integrated SPARQL endpoint service using *Jena Fuseki* allows performing queries and exporting the results in different formats. This enables users to test whether the vocabulary meets their requirements. Additionally, it checks for the existence of files with the extension *.rq* defining queries. All files found are uploaded to this service by taking the file name as the query name, and the content of the file as the query. Furthermore, we developed a tool that automatically executes queries for constraint violation checking. Some examples

[9] https://github.com/schemaorg/schemaorg/, https://github.com/dgarijo/Widoco.
[10] http://www.w3.org/TR/swbp-vocab-pub/.

Table 1. Examples of predefined queries for constraint checking.

Query name	Expected value	Required
At least one *owl:Ontology* needs to be defined	isNotEmpty	Mandatory
Two resources should not have the same *rdfs:label*	isEmpty	Mandatory
Two resources should not have the same *rdfs:comment*	isEmpty	Mandatory
All resources should have *rdfs:label* and *rdfs:comment* in English	isEmpty	Optional
All resources must not have literals with “foo bar”, “lorem” or “ipsum”	isEmpty	Mandatory
All resources should have *rdfs:label* different from *rdfs:comment*	isEmpty	Optional
All resources should have *rdfs:comment* in different languages	isEmpty	Optional
All skos:Concepts should be *skos:inScheme*	isEmpty	Mandatory
All skos:Concepts should have a *skos:broader* statement	isEmpty	Optional

of these predefined queries, that can be easily changed or extended, are listed in Table 1. Whenever the value that is returned after executing the corresponding SPARQL query does not match the value in the “Expected Value” column, this is an indication for constraint violation. The results of this validation process is reported in HTML format. Listing 1 depicts the SPARQL query that checks for missing *rdfs:label* and *rdfs:comment* in English.

```
SELECT DISTINCT ?r WHERE { ?r rdf:type ?type .
        MINUS { ?r rdf:type skos:Concept. }
        MINUS { ?r rdf:type skos:ConceptScheme. }
        OPTIONAL { ?r rdfs:label ?label .
          FILTER( (STRLEN(?label) > 0) && langMatches( lang(?label),'en' ))}
        OPTIONAL { ?r rdfs:comment ?comment .
          FILTER((STRLEN(?comment) > 0) && langMatches( lang(?comment),'en'))}
          FILTER ( !bound(?label) || !bound(?comment) )
    } ORDER BY ?r
```

Listing 1. Resources should have at least one English *rdfs:label* or *rdfs:comment*.

4.4 Deployment

We deploy the VoCol implementation as VirtualBox and Docker virtual machine images, which can be installed with little effort. The VoCol environment thus works as an isolated solution without affecting the rest of the physical machine. This ensures high portability, allowing the administrator to easily start, stop, move, or share it. With a few additional steps, the VoCol environment can be installed and configured on a clean web server. All implementation details are available on the VoCol website[11].

[11] http://vocol.visualdataweb.org.

5 Evaluation

We are currently applying VoCol in an industrial use case to evaluate its usefulness and effectiveness in a real-world setting. Furthermore, we conducted a qualitative user study to get additional insights into the usefulness and usability of VoCol.

5.1 Industry Application

VoCol is currently applied in an industrial use case to develop vocabularies for describing formally the assets of an enterprise, including how they relate to each other. All of these vocabularies, except the developed *rami* vocabulary[12], are the intellectual property of the industrial partner. We are restricted in the information we can provide here, but would like to share at least some experiences and insights.

A group of seven people contributed in parallel to the development of the vocabularies. While the knowledge engineers conducted most of the formalization, the domain experts participated by creating issues. In total, 46 issues were created ranging from proposals to add, modify, or remove vocabulary terms to VoCol environment issues, such as bug fixes and feature requests. The developed vocabularies currently comprise 151 classes, 93 object properties, 225 datatype properties, and 79 instances.

The loose coupling characteristic of VoCol allowed us to integrate a new component for defining and establishing R2RML mappings between the developed vocabularies and legacy data sources of the industrial partner. By doing so, users were able to execute queries against the legacy systems and receive the results in various representation formats, such as tabular, pie, and bar charts, etc.

VoCol provides very useful and effective support in this use case according to the informal feedback of the involved stakeholders. In particular, the different views on the vocabulary provided by VoCol are considered to be very helpful in getting a better understanding and exploring the state of the art. The easy and comfortable access to all services via one integrated web interface was praised by all stakeholders.

Despite the benefits of VoCol for this use case, one of the drawbacks that we experienced is the lack of a simple form-based editing of vocabulary terms. This prevented domain experts from contributing their ideas directly to the development, but required the continuous involvement of knowledge engineers.

5.2 User Study

We conducted a qualitative user study of VoCol under controlled conditions using the *Concurrent Think Aloud* (CTA) method: Participants were observed and asked to verbalize their thoughts while performing the given tasks [18]. At the beginning of each session, the interviewer gave a general introduction into

[12] http://w3id.org//i40/rami.

VoCol. The interaction with the system as well as comments and suggestions were recorded for later analysis. After completing the tasks, participants had a discussion with the interviewer about their experiences and any difficulties they faced. To measure the usability and ease of use, participants were asked to fill a questionnaire at the end of the interview.

Participants: To ensure that participants represent as closely as possible the targeted user group of the VoCol system, we chose twelve users with different levels of expertise, ranging from basic vocabulary modeling experience to more advanced expertise in knowledge conceptualization and representation.

Tasks and Questionnaire: We designed a set of tasks that comprised all activities of the round-trip development described above (cf. Sect. 2): Starting from the modeling activity, the first task was to define several classes with various numbers of properties. The next task was concerned with the population of the vocabulary, in which users had to create instances based on the defined classes. The tasks were performed on the user machine by committing all changes to the local repository first and later pushing those changes to the remote repository. The SPARQL endpoint was used to execute test queries verifying whether the developed vocabulary met certain criteria. All functionalities provided by VoCol, including the syntax validation before commit and after push events to the remote repository, documentation generation, visualization, etc. were covered in the user study.

In addition, we asked the participants to fill an electronic post-study questionnaire composed of two main sections. The first section contained the USE Questionnaire[13], which uses five-point Likert scales for rating, ranging from 1 (strongly disagree) to 5 (strongly agree). We evaluated four usability dimensions: (1) usefulness; (2) ease of use; (3) ease of learning; and (4) satisfaction. To get more insights into specific areas, we defined three additional questions in the second section of the questionnaire. With these questions, we aimed to get the participants' opinion about: (1) the importance of the individual services integrated into VoCol; (2) negative and positive aspects of the system through an open response question; and (3) possible services to be integrated in the future. The evaluation material is available online.[14]

Results: We obtained the evaluation results by observation, discussions at the end of each session, and the post-study questionnaires. The following are some of the findings that we derived from the analysis of the observation notes and discussions:

– Participants with prior knowledge about VCS, especially with Git, found VoCol very easy to learn and use.
– A few participants expected to see provenance metadata in the browser, i.e., the date and author for each term added to the vocabulary.
– The instant syntax-checking and auto-completion feature of the TurtleEditor was considered very helpful by the majority of the participants.

[13] http://hcibib.org/perlman/question.cgi?form=USE.
[14] https://figshare.com/articles/VoCol_Evaluation_Material/3438371.

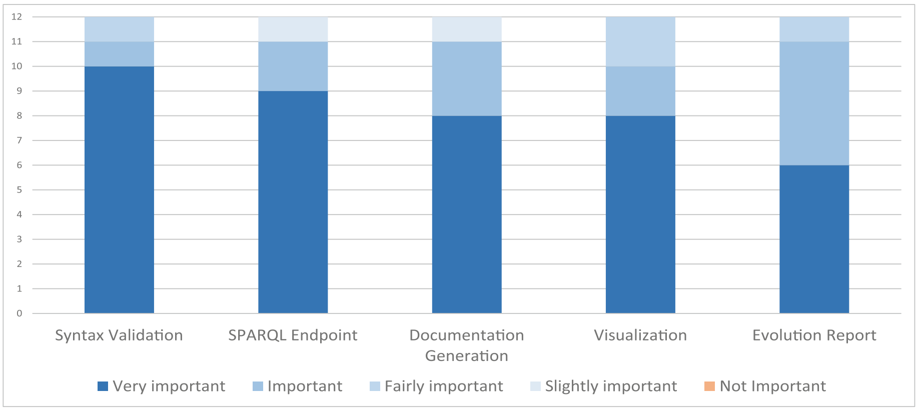


Fig. 4. Importance of the VoCol services according to the study participants.

The results from the USE questionnaire showed that the responders rated their experience with VoCol very high. The average scores received by each dimension are as follows: *usefulness = 4.34*, *ease of use = 3.97*, *ease of learning = 4.35*, and *satisfaction = 4.31*. These scores indicate a high usability of VoCol (nearly all scores are > 4) and correlate with the oral feedback of the participants that VoCol is “easy to learn and use”, as well as the informal feedback of the stakeholders form the industrial use case that VoCol provides “very useful and effective support”.

Figure 4 shows that each of the services provided by VoCol is of high relevance to the study participants. For instance, 10 of the 12 participants consider *syntax validation* a *very important* service, while the scores for the other services are only slightly lower. Some interesting suggestions made by the participants were: (1) creating a possibility for dynamically adding and removing tools from the user interface; and (2) automatic recommendation of similar vocabularies (e.g., using the LOV[15] API).

6 Related Work

Vocabulary development is an active research topic in the Semantic Web community [15]. One area of research is concerned with the development of web applications that offer low-barrier access to vocabulary development. A well-known approach in this area is *WebProtégé* [22], which is a lightweight version of the Protégé desktop editor. It offers change tracking and collaboration features to support the distributed development of vocabularies, and comes with a customizable user interface that can be adapted to different expertise levels of the users. *VocBench* [21] is a web application targeted at editing SKOS and SKOS-XL thesauri. It supports the workflow management, validation, and publication of vocabularies, and provides a full history of changes as well as a SPARQL query

[15] http://lov.okfn.org/dataset/lov/vocabs.

endpoint. VocBench implements the separation of responsibilities through a role-based access control mechanism, checking user privileges for the different tasks of thesauri editing. *SOBOLEO* [24] also fosters the collaborative editing of SKOS thesauri. It provides a specialized browser to navigate and change the taxonomy and a semantic search engine for annotating web resources. SOBOLEO is used in the domain of social networks and offers tag recommendations for describing people based on existing vocabularies. *TopBraid Enterprise Vocabulary Net* (TopBraid EVN)[16] is a proprietary tool to ease the collaborative creation of SKOS taxonomies and ontologies. It incorporates change audits, role management, concept search capabilities as well as data quality rules to check SKOS and OWL constraints. Moreover, it enables the creation of hierarchy reports through graphical user interfaces. *MoKi* [2] is a collaborative MediaWiki-based tool to support the ontological modeling tailored for business processes. MoKi associates a wiki page, containing both unstructured and structured information, to each entity of the ontology and process model. *PoolParty* [19] provides a web interface for building and managing SKOS thesauri. A user-friendly GUI facilitates the participation of domain experts. It also allows to extract relevant information from external Linked Data sources. *TemaTres*[17] is a web application optimized for SKOS thesauri. It includes an API to access the latest version of the vocabulary, a WYSIWYG editor, and extensive quality assurance support.

The main objective of all these tools is to support the collaborative web-based editing of vocabularies. Although they contain many interesting features for vocabulary development, they are not focused on reusing existing VCSs as a core component of the vocabulary development process. More closely related to VoCol are approaches that aim to extend VCSs with additional features dedicated to vocabulary development.

SVoNt [12] proposes to use Apache Subversion (SVN) as a VCS for the versioning of ontologies. SVoNt uses a separate server to store conceptual changes between different versions of ontologies. These versions are generated as a result of a *diff* operation between the modified and base ontology. SVoNt supports conflict detection and resolution by comparing the structure and semantics of the ontologies. *Ontoology* [1] is a tool for vocabulary development based on Git, similar to the presented VoCol implementation. It generates a documentation using *Widoco*, while an ontology pitfalls report is provided based on the *OOPS* service[18]. Ontoology uses AR2DTool[19] for creating class and taxonomy diagrams. The generated artifacts can become part of the repository after a *pull request* is performed. However, providing a user-friendly client which hides the complexity of the version control system is not in the focus of these works. Thus, these systems are rather suited for ontology development projects that involve purely users with a strong technical background. Furthermore, they do not provide a set of services that is as comprehensive and integrated as that of VoCol.

[16] http://www.topquadrant.com/products/topbraid-enterprise-vocabulary-net/.
[17] http://www.vocabularyserver.com.
[18] http://oops.linkeddata.es.
[19] https://github.com/idafensp/ar2dtool.

7 Conclusions and Future Work

We have presented VoCol, an integrated environment for distributed development of vocabularies based on version control systems. We have defined *distributed vocabulary development* as the process of identifying the main terms and concepts among the involved stakeholders and finding a consensus between them. We argue that the development of an effective and efficient environment for distributed collaboration is the main challenge in this context. The presented VoCol environment supports the identified requirements by extending the functionality of plain version control systems with external tools via the webhook mechanism.

We implemented VoCol on the basis of the widely used VCS Git. Tasks such as content negotiation, documentation and visualization generation, as well as evolution tracking are performed in a fully automated way. In addition, a querying service, synchronized with the latest version of the vocabulary, enables users to execute SPARQL queries. The VoCol environment is easily expandable with other tools to provide additional functionalities. The current implementation of VoCol is tailored to small to medium size vocabularies. However, it can be adjusted for various scenarios by replacing its components with adequate alternatives.

For future work, we plan to implement VoCol also for other VCSs, such as Subversion and Mercurial. Furthermore, we envision an automatic population service that creates data according to the defined terminology of the vocabulary. Finally, we plan to provide VoCol as a service where users can simply subscribe their repositories and benefit from all functionalities.

Acknowledgments. This work has been supported by the German Federal Ministry of Education and Research (BMBF) in the context of the projects *LUCID* (grant no. 01IS14019C), *SDI-X* (no. 01IS15035C) and *Industrial Data Space* (no. 01IS15054).

References

1. Alobaid, A., Garijo, D., Poveda-Villalón, M., Santana-Perez, I., Corcho, Ó.: Ontoology, a tool for collaborative development of ontologies. In: ICBO 2015, CEUR-WS, vol. 1515 (2015)
2. Ghidini, C., Rospocher, M., Serafini, L.: Moki: a Wiki-based conceptual modeling tool. In: ISWC 2010 Posters and Demos, CEUR-WS, vol. 658 (2010)
3. Giuliano, C., Gliozzo, A.M.: Instance-based ontology population exploiting named-entity substitution. In: COLING 2008, ACL, pp. 265–272 (2008)
4. Gracia, J., Montiel-Ponsoda, E., Cimiano, P., Gómez-Pérez, A., Buitelaar, P., McCrae, J.: Challenges for the multilingual web of data. J. Web Semant. **11**, 63–71 (2012)
5. Grüninger, M., Fox, M.S.: Methodology for the design and evaluation of ontologies. In: IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing (1995)
6. Halilaj, L., Grangel-González, I., Coskun, G., Lohmann, S., Auer, S.: Git4Voc: collaborative vocabulary development based on git. Int. J. Semant. Comput. **10**(2), 167–192 (2016)
7. Halilaj, L., Grangel-González, I., Vidal, M.E., Lohmann, S., Auer, S.: Proactive prevention of false-positive conflicts in distributed ontology development. In: IC3K 2016, to appear

8. Heath, T., Bizer, C.: Linked data: evolving the web into a global data space. Synth. Lect. Semant. Web: Theor. Technol. **1**(1), 1–136 (2011)
9. Kaur, A., Mann, K.S.: Component based software engineering. Int. J. Comput. Appl. **2**(1), 105–108 (2010)
10. Lohmann, S., Link, V., Marbach, E., Negru, S.: WebVOWL: web-based visualization of ontologies. In: Lambrix, P., Hyvönen, E., Blomqvist, E., Presutti, V., Qi, G., Sattler, U., Ding, Y., Ghidini, C. (eds.) EKAW 2014. LNCS (LNAI), vol. 8982, pp. 154–158. Springer, Heidelberg (2015). doi:10.1007/978-3-319-17966-7_21
11. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Visualizing ontologies with VOWL. Semant. Web **7**(4), 399–419 (2016)
12. Luczak-Rösch, M., Coskun, G., Paschke, A., Rothe, M., Tolksdorf, R.: SVoNt: version control of OWL ontologies on the concept level. In: AST 2010, GI, pp. 79–84 (2010)
13. Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A framework for ontology evolution in collaborative environments. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 544–558. Springer, Heidelberg (2006)
14. Noy, N.F., Tudorache, T.: Collaborative ontology development on the semantic web. In: AAAI Spring Symposium: Semantic Web and Knowledge Engineering, pp. 63–68 (2008)
15. Palma, R., Corcho, O., Gómez-Pérez, A., Haase, P.: A holistic approach to collaborative ontology development based on change management. J. Web Semant. **9**(3), 299–314 (2011)
16. Petersen, N., Coskun, G., Lange, C.: TurtleEditor: an ontology-aware web-editor for collaborative ontology development. In: ICSC 2016, pp. 183–186. IEEE (2016)
17. Ren, Y., Parvizi, A., Mellish, C., Pan, J.Z., van Deemter, K., Stevens, R.: Towards competency question-driven ontology authoring. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 752–767. Springer, Heidelberg (2014)
18. Russo, J., Johnson, E., Stephens, D.L.: The validity of verbal protocols. Mem. Cogn. **17**, 759–769 (1989)
19. Schandl, T., Blumauer, A.: PoolParty: SKOS thesaurus management utilizing linked data. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 421–425. Springer, Heidelberg (2010)
20. Simperl, E., Luczak-Rösch, M.: Collaborative ontology engineering: a survey. Knowl. Eng. Rev. **29**(01), 101–131 (2014)
21. Stellato, A., Rajbhandari, S., Turbati, A., Fiorelli, M., Caracciolo, C., Lorenzetti, T., Keizer, J., Pazienza, M.T.: VocBench: a web application for collaborative development of multilingual thesauri. In: Gandon, F., Sabou, M., Sack, H., d'Amato, C., Cudré-Mauroux, P., Zimmermann, A. (eds.) ESWC 2015. LNCS, vol. 9088, pp. 38–53. Springer, Heidelberg (2015)
22. Tudorache, T., Nyulas, C., Noy, N.F., Musen, M.A.: WebProtégé: a collaborative ontology editor and knowledge acquisition tool for the web. Semant. Web **4**(1), 89–99 (2013)
23. W3C: Vocabularies (2015). https://www.w3.org/standards/semanticweb/ontology
24. Zacharias, V., Braun, S.: Soboleo - social bookmarking and lighweight engineering of ontologies. In: CKC Workshop at WWW 2007 (2007)
25. Zaikin, I., Tuzovsky, A.: Owl2vcs: Tools for distributed ontology development. In: OWLED 2013, CEUR-WS, vol. 1080 (2013)