VocBench 3: a Collaborative Semantic Web Editor for Ontologies, Thesauri and Lexicons

Abstract. VocBench is an open source web platform for the collaborative development of datasets complying with Semantic Web standards. Since its public release – five years ago – as an open source platform, VocBench has attracted a growing user community consisting of public organizations, companies and independent users looking for open source solutions for maintaining their thesauri, code lists and authority resources. The focus on collaboration, the differentiation of user roles and the workflow management for content validation and publication have been the strengths of the platform, especially for those organizations requiring a distributed, yet centrally controlled, publication environment. In 2017, a new, completely reengineered, version of the system has been released, broadening the scope of the platform: funded by the ISA2 programme of the European Commission, VocBench 3 offers a general-purpose collaborative environment for development of any kind of RDF dataset (with dedicated facilities for ontologies, thesauri and lexicons), improving the editing capabilities of its predecessor, while still maintaining the peculiar aspects that determined its success. In this article, we review the requirements and the new objectives set for version 3, and then introduce the new characteristics that were implemented for this new incarnation of the platform.

Keywords: Collaborative Editing, Ontologies, Thesauri, Lexicons, OWL, SKOS, OntoLex

1. Introduction

In 2008 the group for Agriculture Information Management Standards of the Food and Agriculture Organization of the United Nations (FAO, http://www.fao.org/) developed a collaborative platform for collaboratively managing their Agrovoc thesaurus [1]. The so-called “Agrovoc Workbench” soon met the interest of other FAO departments and several other organizations interested in open source solutions for collaborative thesauri development.

Baptized with a new name: “VocBench” – suggesting a more general environment for thesauri management – the platform later had been strongly re-engineered in the context of a collaboration between FAO and the ART group of the University of Rome Tor Vergata (http://art.uniroma2.it). The result of this collaboration, VocBench 2 [2], released in 2013, had been rethought as a fully-fledged collaborative platform for thesaurus management, freely available and open-sourced, offering native RDF support for SKOS [3] and SKOS-XL [4] knowledge organization systems [5], while retaining from its original version the focus on multilingualism, collaboration and a structured content validation & publication workflow. The system, in its first release, was the result of a six-months development effort aiming at replacing the original VB1 backend for RDF, rewiring the web platform to the RDF Management framework Semantic Turkey [6, 7], already developed by the ART Group.

The possibility for project administrators to define roles with very specific capabilities and to assign them...
to different users according to their proficiencies and authorizations, together with the publication workflow where dedicated users could supervise the work of others and accept their modifications, have been the very strengths of the platform, especially for those organizations requiring a collaborative yet centrally controlled publication environment. Consistently with this controlled approach, what some other users felt as mostly missing from the system was more freedom on data shaping, longing for unrestricted capabilities for editing data at its very core, as in triple-oriented RDF editing environments. This larger freedom would also reflect in the possibility to customize the models, going beyond plain SKOS/SKOS-XL modeling, which is sometimes a must for users dealing with complex, yet still KOS-like, resources.

There is indeed a trade-off in modeling systems between flexibility and control, which is non-trivial to overcome. Controlling the actions that users can perform implies recognizing them as high-level performative, associated to re-defined modeling graph patterns and described by rich metadata that can be used to identify them, to authorize their invocation by classes of users, to properly store them in validation stacks, etc… This complex castle is thus usually not meant to be maintained brick-by-brick: the various aspects described above and the dependencies among them are easier to be managed by identifying clear “first-class citizens” (entities recognized by the system and on which operations can be performed) and a pre-defined set of operations (possibly, non-overlapping) on them. The final touch is to implement this higher layer of representation in terms of lower-level aspects of the model/technology (i.e. triples in RDF).

The above considerations are at core motivations behind the impetus that led to the design and development of a new incarnation of the system, VocBench 3 (or, simply, VB3). The VB3 project was planned to overcome the above limitations, while broadening the original scope of the platform to a general-purpose collaborative environment for development of SKOS thesauri, OWL ontologies and RDF datasets in general. The VocBench 3 project is funded by Action 1.1 of the ISA2 Programme of the European Commission for “Interoperability solutions for public administrations, businesses and citizens” (https://ec.europa.eu/isa2/). The action is managed by the Publications Office of the European Union (https://publications.europa.eu/). VB3 has been developed in close collaboration with the ART group of the University of Rome Tor Vergata, the same group that contributed to the development of the second version of the platform.

VocBench 3 has been released to the public on September 2017, under a BSD 3-clause license1. Since then, two revised versions of the system have been released. The VocBench site (http://vocbench.uniroma2.it/) contains documentation, download links and other references for the new version, while keeping analogous material for the legacy VocBench 2, which is still adopted by many organizations needing time and effort to perform the switch to the new version. A second development iteration is being carried on, started after the first release of VB3 and terminating on July 2018 with VB3 v.4.02.

In this article, we review the original requirements that drove the development of the platform and introduce the new objectives set for VB3. We then describe and discuss the new features and architectural improvements that have been implemented to meet the goals for this next iteration of the platform. Our aim is thus to highlight the improvements over VB2, while we refer the reader to [2] for a general introduction to the system and for a comparative analysis of VB with related works.

2. Requirements

In this section, we list the foundational stones upon which VocBench 3 was developed, in terms of the requirements to be met. These requirements include the set of requirements originally put together for the development of the original VocBench platform and of its second iteration (R1-R7 in the list below). These requirements have been reassessed and reformulated, accounting for the widened scope of the platform and its improved editing capabilities. Other requirements (R8-R15) have also been added, following proposals by its developing team and requests by stakeholders presented in dedicated stakeholder meetings.

R1. Multilingualism. Properly characterizing knowledge resources in different (natural) languages is fundamental. This especially holds for thesauri, due to their use in Information Retrieval, though the overall importance of elaborated lexicalizations is indeed a different project from VB2, so that the first release has been marked as version 1.0 of VB3.

1 https://opensource.org/licenses/bsd-3-clause
2 VB3 adopts Semantic Versioning (http://semver.org). In that context the major release number has specific semantics. We thus opted for considering the “3” (as in VB3) as part of the name (VB3).
progressively gaining momentum, thanks to data publication initiatives such as the Linguistic Linked Open Data (LLOD: http://linguistic-lod.org/) and to models for Ontology-Lexicon interfaces, such as lemon (lexicon model for ontologies) [8] and its most recent specification OntoLex-Lemon [9], realized in the context of the homonymous W3C community group: https://www.w3.org/community/ontolex/.

R2. Controlled Collaboration. One of the keystones of the system is to enable collaboration on a large scale: several, distributed users have to be able to collaborate remotely on the same project. Opening up to communities is important, though the development of authoritative resources demands for the presence of some control to be exerted over the resource lifecycle: for this reason, users must be granted different access levels, and some of them should be granted the possibility to validate other users’ work before it is finally committed to the dataset.

R3. Data Interoperability and Consistency. Interoperability of several resources critically depends on data integrity and conformance to representation standards. However, flexible models such as SKOS translate to underspecified possibilities on the one hand, and formal constraints beyond the expressiveness of OWL on the other. Additionally, the increase in the offer of – often overlapping – standards in the RDF family of languages for the Semantic Web resulted in the necessity for systems to be flexible enough to properly read and manage all of them. It is thus important that VocBench enforces a consistent use of these models, by preventing the editors from generating invalid data, and by providing “fixing facilities” for spurious data acquired from external sources. Finally, support for alignment to other datasets is also an interoperability must for the Linked Data World.

R4. Software Interoperability/Extensibility. The system should be able to interact with (possibly interchangeable) standard technologies in the RDF/Linked Data world, with the possibility to surf linked open data on the Web, accessing SPARQL endpoints, resolving RDF descriptions through HTTP URIs etc., as well to import/export data through standard Graph Store APIs and the like.

R5. Data Scalability. The system must deal with (relatively) large amount of data, still offering a friendly environment. This is particularly true for some thesauri as well as for most lexicons. The User Interface must thus consider this requirement by appropriately subdividing data loading into subsequent requests and implementing dedicated solutions for large results.

R6. Under-the-hood data access/modification. While a friendly UI for content managers/domain experts is important, knowledge engineers need to access raw data beyond the usual front-ends, as well as to benefit from mass editing/refactoring facilities.

R7. Adaptive Context and Ease-of-use. In migrating from the first VocBench to its second version, it was mandatory that different users, ranging from ordinary editors to system administrators, shared an easy and comfortable user experience. The new VB3 should provide an even smoother experience, with very low installation requirements and an as-short-as-possible time-to-use. Whether (and proportionally if) the user is an administrator configuring the system, a project manager configuring a project, a user requesting registration and connection to a given project, or a new user willing to test the system as a desktop tool without settings and configuration hassle, the platform should respond adaptively to their needs.

R8. RDF Languages Support. Differently from both its predecessors, dealing with thesauri only, VB3 has to offer native support for SKOS (SKOS-XL) thesauri, OWL ontologies, and RDF datasets in general. As of a second development iteration started in late 2017, support for OntoLex-Lemon lexicons and for ontologylexicon interfaces has been introduced as a further requirement for the system [10].

R9. Maintainability (Architecture and Code Scalability). In special mode, the ability to meet new requirements, cope with changed environments and make future maintenance easier. A weak spot of VB2, VB3 aims to achieve high levels of architecture/code scalability. In VB3 it is mandatory to be able to add new services, functionalities, plugins etc... without the fabric of the system being altered or too much effort being required to align these new elements with all the characteristics of the system, such as validation, history management, roles and capabilities.

R10. Full Editing Capability (RDF Observability and Reachability). Any complex RDF construct should always be inspectable and modifiable by users (providing they have the proper authorization) even in its finer details. While the platform can provide high-level performatives for conveniently creating/modifyng complex descriptions of resources according to pre-defined modeling design patterns (i.e. by using RDF graph patterns with variables being instantiated upon usage), the user should never be prevented from inspecting/altering these elements.

R11. Provenance. Actions in VB3 should be handled as first-class citizens themselves, being identified and
qualified by proper metadata, logged in a history with information about which user performed an action, when they did it, which parameters have influenced its performance, etc... Metadata answering to the five “Ws” (with the possible exception of the “why”) should provide all information for tracking the origin of an action.

R12. Versioning Support. Besides the history and validation mechanisms, providing triple-grained information about actions enriched with metadata about provenance, it should be possible to take static, periodic, snapshots of the dataset.

R13. Dataset-level Metadata Descriptions. For the Semantic Web to fully achieve its vision, linked open data has to speak about itself [11]. This means not only having data modeled according to well-known shared vocabularies, but to be able to grasp meaningful information about a dataset without having to dig into its content. Edited datasets should be coupled with resuming information about their characteristics, that can be published together with them in order to be properly consumed by clients.

R14. Customizable UI. UIs merely based on ontology description are limited to the analysis of the axiomatic description of the resources they show and of their types, ignoring possible desiderata of the user. VB3 should allow users to represent the information that they want to specify at resource creation, per resource type, so that it will be prompted to the user. Connected technical aspects, such as proper transformation of the user input into serializable RDF content, should also be tackled.

R15. Everything’s RDF. VB2 used a relational database to store user & project management information, history & validation information. Conversely, VB3 should follow a more uniform approach, adopting RDF for virtually any information that needs to be stored.

In next two sections, related to architecture and features of the system, we discuss the main characteristics introduced in the new edition of the software that allowed meeting the aforementioned requirements.

3. Architecture

VB3 is based on a classical three-tier architecture (Figure 1), structured through a presentation layer, a service layer and a data layer. This already represents a difference with respect to VocBench 2, as a bird’s view over the main layers of VB3 immediately reveals a more streamlined organization than the one of its predecessor. Indeed, as a result of a quick merge between FAO’s collaborative platform VocBench and ART’s Semantic Turkey (also shortened as ST), VocBench 2’s business logic was split among the collaboration-oriented characteristics, user and project management, history and validation features that were inherited from the original VB1 and the native RDF management services introduced by ST. In VocBench 3, the web application (completely re-developed using the Angular framework 3) is now merely a user frontend for the services of Semantic Turkey, which has then evolved into a fully-fledged collaborative environment for RDF management. We can say, in a sense, that VocBench 3.0 is Semantic Turkey on steroids, with a new grown set of feathers.

3.1. Abandoning the RDF abstraction layer of VB2

Another immediately recognizable change with respect to the architecture presented in VocBench 2 (and in Semantic Turkey in the specific) is the strict adoption of the RDF4J 4 framework. Semantic Turkey’s RDF API used in VB2 were based on OWLART 5, an abstraction layer supporting access – through dedicated implementations of the abstraction – to different RDF middlewares, such as Jena 6 [12] and Sesame [13]. That choice was dictated by the absence of an official RDF API for Java, resulting in an effort to bring in as many triple stores as possible through different middlewares. However, after years of maintenance, experience unveiled the limitations of this choice: the various middlewares have evolved, with different paces and on different directions (i.e. focusing on supporting different emerging standards of the RDF family); keeping the pace in adapting the OWLART implementations with the main RDF frameworks required a considerable effort, the results of which are in any case limited to a least common denominator among all of them. Additionally, interoperability cannot even be taken for granted by this approach. Even within a same middleware, triple stores might retain critical differences in the way they organize content (e.g. where the inferred triples are stored and how they can be retrieved) and (thus, but not only for that) in the responses they provide when replying to same API calls. Also, the strategies of SPARQL query resolvers and optimizers may vary dramatically across diverse triple

3.2. Tier architecture

http://rdf4j.org/
With the advent of RDF4J as the new version of Sesame, moved under the stewardship of the Eclipse Foundation, we decided time was mature for taking a stand and adopt that RDF framework as the official RDF API for Semantic Turkey. Constraining the choice to a specific framework, other than sparing us the effort for developing our own middleware, offered us a plethora of framework-specific solutions that we

stores, making it difficult to predict which shape of a SPARQL query will behave as expected under all conditions. Fighting these idiosyncrasies is not an easy war, especially in an editor that must guarantee large flexibility and that cannot perform any content-oriented optimization (as the datasets it manages are not known a-priori).
could exploit, such as embracing RDF4J’s sail-mechanism for developing storage-side components for intercepting effectively-modified triples, which can be seen represented under the local and remote triple stores in the architecture view (see section 3.2 and section 4.4).

3.2. Extendible Architecture

The adoption of the OSGi (formerly Open Service Gateway initiative) service framework allows for dynamic plugging of extensions. We have thus developed a very rich mechanism for describing extensions: in Semantic Turkey there is a general concept of component, an object that can be identified in the system, configured, and scoped (to four domains, i.e. the whole SYSTEM, a USER, a PROJECT or spaces defined by <USER,PROJECT> pairs). Extension Points are components defining extensible functionalities, such as data loading/deployment, rendering of RDF resources on the UI, connection to and interaction with collaboration platforms etc. For each functionality, an implementing class defining an extension point and an interface for the associated extension are provided by the system. Developers can then provide a pluggable implementation of an extension point by creating:

- An implementation of the extension interface associated to the functionality (this will also automatically associate the extension to the extension point)
- A class implementing the ExtensionFactory interface, in charge of creating extension instances on demand by the system
- Any needed settings for the configuration of the extension. The settings can be modeled through a rich type system, which includes basic datatypes, RDF term types and other specific object types that can be addressed by VocBench.

While there is currently no native extension mechanism for web interfaces built through Angular technology, the UI of VocBench is informed by the extension point mechanism and supports extensions with automatically built forms for configuration of settings, selectors for choosing the implementation to choose etc...

Currently, the following extension points have been included in the system (these include functionalities that will be released with the forthcoming version 4.0 of the system):

- **CollaborationBackend**: support for different collaboration platforms. Currently an instance for JIRA\(^7\) is available.
- **DatasetMetadataExporter**: support for the representation and export of metadata about the edited dataset, according to different vocabularies (see section on Metadata Export)
- **Input/Output related extension points**:
  - **Deployer**: a deployer allows VocBench to deploy data to a certain destination. A general HTTP Deployer allows VB to deploy data to any HTTP service accepting data over HTTP. A more specific GraphStore API deployer allows users to directly publish a thesaurus on a target repository in a GraphStore compliant triple store.
  - **Loader**: the dual extension point to the Deployer, allowing VB to load data from different sources
  - **RDFLifter**: an RDF lifter makes VB compliant with different data formats, being in charge of transforming their data into RDF before it is fed to the platform
  - **ReformattingExporter**: dual to the RDF lifters, Reformatting Exporters allow VB to transform exported data to other non-RDF formats
  - **RepositorySourcedDeployer**: a deployer that directly deploys triples to a triple-oriented information destination (usually a triple store).
  - **RepositoryTargetingLoader**: a loader interacting with triple-oriented information sources (usually triple stores) in order to import triples directly from them
  - **StreamSourcedDeployer**: a deployer able to export the data (previously serialized/reformatted by a ReformattingExporter) to any stream-based information host
  - **StreamTargetingLoader**: a loader able to import data from any stream-based information source, before handing it to an RDFLifter for conversion into RDF triples.
- **RDFTransformer**: previously known as export filters and acting in the middle of a data export chain, we changed the name to transformers in that these components can not only filter, but also create new data, and can be placed both in export and import procedures. In an export chain, a transformer can indeed be used to create new data, such as publication metadata or, for instance, redundant data such as materialization of inferred triples. The general

\(^7\)https://www.atlassian.com/software/jira
applicability of RDF transformers made them a re-usable component in different workflows, such as the export of the results of SPARQL graph queries, where the result can be further manipulated. For instance, a general SPARQL query could extract all alignments available in a dataset, while a chained RDF transformer could filter only those alignments referring to DBpedia [14]. The alignment export query can thus be reused across different cases.

- **RenderingEngine**: the engine that produces intelligible labels for each resource to be represented in VocBench. Each rendering engine provides a SPARQL fragment that is usually connected to the queries of the various resource retrieving services.
- **RepositoryImplConfigurer**: this consists in configuration templates for different kind of triple stores. Currently, implementations for the various storage solutions of RDF4J and for GraphDB Free and SE sails are available.
- **SearchStrategy**: a search component that can provide different strategies for the search of resources.
- **URIGenerator**: a pluggable component for the automatic generation of URIs. While the URIGenerator packed with VocBench is highly configurable, it is possible to develop specific generators that comply with custom patterns that cannot be generated by it.

### 3.3. Project and User Management

As anticipated in the description of the architecture, Project and User Management have been completely rewritten as part of the Semantic Turkey framework. The system offers an abstract representation of the core entities managed: users, projects, system properties etc... and API for storing/retrieving them, so that different implementations can be provided.

The default implementation represents another aspect which is streamlined (yet no less elaborated) with respect to VB3’s predecessors: it is completely file-based. As a consequence, the relational DB previously used for system administration (and for metadata representation, which is not completely represented in RDF, see section 4.4) is no more necessary. The use of the filesystem is not just a choice to get rid of the relational DB. An accurate organization in the distribution of the descriptors for users, projects, configurations, settings, plugins (these last three can in turn be also scoped differently) and their relationships allows for an easy porting of data across different distributions: it is thus possible to easily transfer all data to another installation of Semantic Turkey, or to separately move users, projects, to decide if to copy or not their settings etc… by performing simple selections on clearly identifiable subdirectories of the directory where ST stores its data.

### 3.4. CODA

Another piece of software developed by the ART Group, CODA® [15] is a is an architecture and an associated Java framework for the RDF-triplification of results from analysis of unstructured content. The purpose of CODA is to support the entire process embracing data extraction and transformation, identity resolution up to feeding semantic repositories with knowledge extracted from unstructured content. The motivation behind CODA lies in the large effort and design issues required for developing RDF compliant knowledge acquisition systems on top of well-established content analytics frameworks such as UIMA® [16] and GATE® [17]. Therefore, CODA extends UIMA with facilities and a powerful language – PEARL® [18] - for projection and transformation of annotated content into RDF.

CODA has been integrated in ST as a multi-purpose knowledge acquisition and triplification component. While its most natural and implied use – acquisition of knowledge from text – has not yet been exploited in the platform, the advantages of having such a component are manifold. CODA has been firstly embedded in the system to power Custom Form (see section 4.1.4), a feature for enabling customized forms for prompting information about arbitrarily complex RDF constructs. Recently, in the course of the second development iteration of VB3, CODA has also been used as the foundation layer for Sheet2RDF® [19], a platform for the acquisition and transformation of spreadsheets into RDF. Sheet2RDF had originally been developed for a past version of ST and has now been re-engineered for the new version, improved and enriched with a new UI specifically developed for VocBench 3.

---

1. [http://art.uniroma2.it/coda/](http://art.uniroma2.it/coda/)
2. [http://art.uniroma2.it/coda/documentation/pearl.jsf](http://art.uniroma2.it/coda/documentation/pearl.jsf)
3. [http://art.uniroma2.it/sheet2rdf/](http://art.uniroma2.it/sheet2rdf/)
4. VocBench 3 New and Improved Features

In this section we list a set of features, functionalities and visual changes that are more evident to the user than the description provided in the architecture. Conformance to and satisfaction of the requirements expressed in section 2 is reported case by case.

4.1. User Interface (UI).

The UI (Figure 2) is the element that vividly marks the difference between VB3 and its predecessors. The user interface has been rebuilt from scratch, by using different technologies – notably Angular (https://angular.io/) in place of Google Web Toolkit (http://www.gwtproject.org/) – and most importantly – reorganizing the user experience. The Data view, letting the user explore the project’s dataset, is the one that mostly represents these changes. The several tabs of VocBench 2 (which inherited and extended the tab-based model of VocBench 1) that populated the “concept details” panel have been replaced with a single component, called resource-view.

4.1.1. The resource-view

The resource-view offers a complete overview of all details of any type of resource. This highlights another difference with VB1&VB2: there are no first-class citizen resources, such as SKOS concepts and SKOS-XL labels; in fact, all resources now can be viewed and edited through the resource-view section. The resource-view is a general component that can be specialized depending on the inspected resource: a few sections are shared among all resources, such as types, listing the rdf:types for the described resource, lexicalizations, listing all the available lexicalizations and properties, listing all general properties not addressed by the other sections, while others are specific to the inspected resource. For instance, the resource-view centered on a concept is composed of the following sections: types, concept schemes of which the concept is a top concept, schemes to which it generally belongs, broader concepts, lexicalizations, notes and the final generic property section, containing all relationships and attributes except those described in the above listed sections. While the mapping of these sections to properties of the core modeling vocabularies is trivial (e.g. types to rdfs:type, schemes to skos:inScheme and...
so on) the sections are however presented with a predicate-object style in order to qualify the predicate, as they might include user-defined or domain-specific subproperties of the above ones. It is worth of note that the general applicability of the resource-view to any resource and the possibility to edit any of their details from it concur to satisfy requirement R10.

4.1.2. Dataset Model and Lexicalization Model

A special mention goes to the lexicalizations section: it represents an abstraction over different kind of properties and modeling patterns for lexicalizations, as it offers specific resolution of their shape, always showing the form of the lexicalization, i.e. merely the label for rdfs:label and for SKOS terminological labels, the skosxl:literalForm for SKOS-XL labels and the ontolex:writtenRep of the canonical form associated to the lexical entry lexicalizing the resource in OntoLex-Lemon. In VB3, the concept of lexical model has been introduced (and separated from the knowledge model, e.g. OWL or SKOS) so that, for instance, it is possible to select SKOS-XL (the lexical extension to SKOS allowing for reified labels) as a lexical model for both OWL ontologies and SKOS thesauri. In SKOS-XL, the lexicalizations section would provide dedicated forms for creating reified labels (with the possibility to specify their URI or to have it assigned automatically) and directly show their skosxl:literalForm in the object field. The object field is also clickable and it opens a resource-view focused on the resource description of the SKOS-XL label. In OntoLex-Lemon, the indirectness is even more evident as one possible path from the lexicalized resource to the shown lexicalization is:

ontolex:isReferenceOf → <ontolex:Sense> → ontolex:isSenseOf → <ontolex:LexicalEntry> → ontolex:canonicalForm → <ontolex:Form> → ontolex:writtenRep → <shown lexicalization>

Many other paths are considered, due to shortcuts on property chains (e.g. ontolex:isDenotedBy chaining ontolex:isReferenceOf/ontolex:isSenseOf) and inverse properties. So, the written representation is shown, but the user can click on it and inspect the full series of linked RDF terms.

This revised model greatly improves requirement R1 by covering not only diverse natural languages, but the different formal languages (and thus, R8 as well) in which the lexical information can be encoded.

The overall data view (Figure 2) is similar in principle to the one of the previous editions, with the data browsing views on the left (previously, only the concept tree), and the description of the selected resource on the right. The section on resource data structures is composed of different tabs, depending on the chosen modeling vocabulary. OWL offers two tabs with a class tree & instance list and a property tree
respective\textsuperscript{13}, while SKOS adds to them a concept tree, a list of schemes and a collection tree (with the tree showing the containment relation between different SKOS collections). OntoLex-Lemon adds tabs for lexicons and lexical entries.

### 4.1.3. Support for Lexicons and Ontology-Lexicon Interfaces

The support for lexicons required, even more than with thesauri, support for a data-scalable user interface (see Figure 3). Modern thesauri are (usually) organized around a hierarchy of concepts that can be easily browsed by progressively expanding the explored branches of the tree. Conversely, lexicons have huge flat lists of entries that are difficult to represent. Additionally, the OntoLex-Lemon model introduces the notion of ontolex:LexicalConcept that represents “a mental abstraction, concept or unit of thought that embodies the meaning of one or more lexical entries”. This type of resource, in lexical databases modeled as WordNet\textsuperscript{14} [20], corresponds to the notion of synset, the semantic index that glues together sets of synonyms (since, the term syn-set), i.e. lexical expressions sharing the same meaning. In Wordnets, the set of synsets is arranged in a structured tree for what concerns synsets related to nouns, but offers a shallow hierarchy (mostly, a horizontal list of elements) for other parts-of-speech, especially for what concerns verbs. We have thus offered different browsing modalities, based on the exploration of the full content, organized according to different data structures (e.g. trees for concepts, classes and resources, lists for schemes and lexicons or single/double-character indexed lists for lexical entries) or a search-based exploration, that uses the search functionality to selectively show matching entries in their associated panel, thus guaranteeing scalable solutions (req. R5) depending on the nature and specific size of each loaded dataset.

VB1&2 showed concepts through their labels in all of the selected languages for visualization. In VB3, an option allows for toggling between the URIs/qnames of the resources, and the string composed by a resource-renderer. Resource renderers provide human-friendly visualizations of resources. Different renderers can be connected to the system as plugins to its dedicated rendering extension point (see Extension Points later on). The default renderer behaves in a way similar to VB1&2, showing labels in all of the selected languages for visualization (again, R1), being configurable in the languages to show.

### 4.1.4. Custom Forms

An important new aspect of the user interface is offered by Custom Forms, a flexible data-driven form definition mechanism that we devised for VocBench, allowing users to perform a declarative specification of the key elements that concur to the creation of a complex RDF resource (satisfying req. R14). In particular, custom forms rely on the combination of the following four key elements:

\textsuperscript{13} A fourth “datatypes” tab will be added with the release of July 2018

\textsuperscript{14} https://wordnet.princeton.edu
• a declaration of the data that is expected to be prompted by the user
• a series of transformations that must be applied to the prompted data in order to produce valid RDF terms
• the organization of the produced RDF terms into meaningful graph patterns, instantiating the template of the resource to be created
• the automatic production of a form layout (see Figure 4) based on the above declarations information that is required for "constructing" a new resource

Custom Forms have been described more in details in [21], which analyzed and evaluated their expressive power by applying them to the use case of representing entities for the OntoLex-Lemon vocabulary. In that work, a subset of the lemon Design Pattern Library [22] was implemented as custom forms, which VocBench exploits to generated a form-based interface for the creation as well as the visualization of diverse lexical entries. Figure 4 reports a form filled with information regarding the entry "director", which is said to denote the property dbo:director in the DBpedia ontology. Furthermore, the syntax-semantics interface is defined by establishing the correspondence between the atom x dbo:director and the stereotypical verbalization y is the director of x.

4.1.5. Administration Pages

Another relevant difference in the UI offer lies in the Project page: system administrators (and other users having equivalent authorizations) can inspect projects in all their details, and easily switch from one to the other, while other users are offered the traditional project list allowing access to only the projects they are registered to. This is particularly convenient for users willing to use VB3 as a desktop tool: in less than a couple of minutes, it is possible to start the system for the first time, configure a simple user with default minimal information and administrative rights, log in and seamlessly use VocBench without the burden of dealing with a complex multiuser collaborative web application (R7).

4.2. Continuous check-on-start

A continuous check-on-start life-cycle also contributes to satisfying requirement R7: VB technically never recognizes itself as installed/deployed, rather at each application startup it checks that the complete set of pre-requisites for a correct start is satisfied. Whenever a new VB version is installed, if new features have been introduced, or mandatory configuration options added, the system will identify these needs and react accordingly, eventually interacting with the user upon necessity

4.3. Controlled Collaborative Editing through Role-Based Access Control (RBAC)

A single installation of VocBench can handle multiple projects, which can also be interlinked for mutual data access (e.g. for purpose of alignment). VocBench promotes the separation of responsibilities through a role-based access control mechanism, checking user privileges for requested functionalities through the role they assume (req. R2). Upon registration, users indicate their personal information and their proficiencies. The proficiencies are merely user declarations and self-assessed skills, so they do not grant any permission per se, but can help administrators and project managers (users with the role of administering a single project) in selecting users to assign to their project, or trivially by simplifying the assignment of capabilities to them by reusing their declared proficiencies as a template/filter. In VB3, we have completely redesigned the mechanism for roles/capabilities. While VB2 had hard-wired roles with predefined and limited editing possibilities, which do not easily scale-up to possible extensions of the system (req. R9), in VB3 we have created a dedicated language for specifying capabilities in terms of area, subjects and scopes E.g. the expression:

auth(rdf(datatypeProperty, taxonomy), R)

corresponds to the requested authorization for being able to read taxonomical information about datatype properties. The 'R' stands for READ, as in the CRUD paradigm, rdf is the area of the requested capability while datatypeProperty and taxonomy define the subject and scope respectively of the capability.

The language is implemented as a series of facts for the Prolog [23] logic programming language. Entailments are guaranteed thanks to axioms expressed by rules written in Prolog (which may be extended by users). One trivial application of entailments is the definition of shortcut expressions: for instance the simple area expression: “rdf” entails any monadic (rdf(_)) or dyadic (rdf(_,_)) expression with the rdf predicate (i.e. implying that the capability expressed by the simple term rdf covers any subject and scope in the area of RDF). Using an interpreted logical language allowed for easily manipulating and validating the same set of expressions on both the server and the client, by using
different technologies. In the server, the computation of expressions in this capability language is performed by the tuProlog [24] engine, a Prolog component written in Java and developed by the University of Bologna. In the web client, jsprolog (https://github.com/Sleepyowl/jsprolog/), a lightweight JavaScript library for interpreting Prolog, performs the same job.

The double implementation of the Prolog engine allows for a double-gate check: the server performs the ultimate checks for authorization (as requests could be performed by bypassing the client or maliciously altering it) while the client one is used to enable visualization/activation of UI elements, depending on the logged user. This way, it is possible to have very dynamic UIs automatically modeled on users’ capabilities, which are ultimately defined in the services, with no redundancy in the code.

New roles can be easily created, and existing ones can be modified, through a dedicated rbac editing wizard (Figure 5). The default policy recognizes typical roles and their acknowledged responsibilities:

- **Administrator**: the sole inter-project role (i.e. the role exists a-priori from projects). The Administrator has, by definition (i.e. evaluation of authorized capabilities is skipped for it), access to all functionalities and configuration options of the system.
- **Project Managers**: project-local administrators. Within the boundaries of the project(s) they have been assigned to with that role, they can do everything: from data and configuration management to assigning users to the project and granting roles to them. Their boundaries are: other projects and system-level settings and configuration.
- **Specific project-local roles**: Ontology editors (authorized to perform changes at the axiomatic level), Thesauri Editors (authorized to work on thesauri without performing OWL editing actions), Terminologists/Lexicographers (authorized to edit lexicalizations, can be limited to edit only certain languages according to their proficiencies), Mappers (authorized to perform alignments only), Validators (can perform validation actions, see “Formal Work-flow Management” section) and finally Lurkers, that can read everything in a project but have no editing authorization.

4.4. Advanced History and Change Tracking mechanism

Both a strength and a weakness in VB2, the Change Tracking mechanism that powered History & Validation was appreciated by many users. However, being
based on a pre-defined set of recognized operations, it severely limited system maintainability (req. R9) and the possibility to perform (req. R6) under-the-hood changes (e.g. through changes brought directly through SPARQL) while keeping a history which is consistent with the status of the dataset. Furthermore, the separate history system (inherited from the original VocBench system) was cumbersome and mostly opaque to analysis, being based on data blobs stored in a relational DB. In VB3 we abandoned the separated relational DB storing user and history data, and implemented, completely in RDF (req. R15), a track-change mechanism working at triple-level. This fine-grained representation was then complemented with rich metadata (R11) about the invoked actions and the context of their invocation. Triples removed/added by each action are reified, grouped around a common resource representing the action that produced the change and stored in a separated (but connected to the project) RDF repository (the support repository) together with the actions’ metadata. The change-tracking mechanism has been implemented as a new sail for the RDF4J framework (http://rdf4j.org/), formerly Sesame [13]. The sail is embedded with the system, but can also be deployed as a pluggable component inside other sail-compliant triple stores (req.R4), such as GraphDB [25].

The design of the history and change tracking mechanism in VB3 was guided by a landscape analysis [26, 27] in which we discussed the nature and the representation of change, reviewed some version control systems for RDF, and delved into the challenges posed by validation.

4.5. More Powerful yet Streamlined Workflow Management

VB2 had a 5-steps publication workflow, clocked by the property “status” (with values: proposed, validated, published, deprecated and proposed_deprecated) and, redundantly, with information stored in the DB about the status of operations to be validated. Also in VB2, the concepts of resource and action were mixed up in the validation procedure, with the status of a resource being affected by the validation (e.g. moving from “proposed” to “validated”), while single changed triples had no trace of their validation status. This follows from the fact that it is not possible to
attach a status to a triple in RDF, if not by reifying the triple. Finally, there is no standard W3C equivalent for the custom “status” property in VocBench, thus reducing this status information about the workflow to something to be removed from the dataset when it gets published.

Benefiting from the new Change Tracking system, we have made things clearer and easier: there is no “status” property anymore, as the workflow is implicitly expressed by the validation mechanism coded into graphs. The added/removed triples are stored in the support repository as described in the previous section, while non-reified “previews” of them are available in separate graphs (staging-add-graph and staging-delete-graph) in the main repository, and the system, being aware of this graph/repository organization, presents them appropriately to the user. In this way, the main graph being written implicitly represents stable information, which does not need to be tagged as “validated”. The distinction between “validated” and “published” has been removed as most users considered this sort of dual validation as useless. Probably, the original intention, dating back to the first VocBench, was to distinguish which resources had been published in an open version of the dataset from those that—though being validated—had never seen the light out of VocBench. However, this distinction has never been put in place in any known user workflow. Finally, the status of deprecation has been represented through the official owl:deprecated property. The status of “proposed deprecated” is also intuitively represented by the need to validate the action for setting the owl:deprecated property to “true”.

4.6. Improved and More Complete Support for SKOS

VB2 had already an advanced support for multiple SKOS schemes. We have improved the management by allowing users to select more schemes for browsing the concept tree and by adopting a combination of conventions and editing capabilities for quickly associating the proper schemes to newly created concepts and collections. Support for SKOS collections and ordered collections has been introduced in the system with dedicated UI views and editing facilities.

4.7. OWL Support

VocBench already allowed for importing ontology vocabularies for modeling thesauri. Now VocBench also supports ontology development (requirement R8), with editing of OWL axioms (using the Manchester syntax for both editing and visualization, see Figure 6) and an almost full coverage of OWL2 expressions.

4.8. SPARQL Querying and Update

A new SPARQL UI, based on YASGUI [28] has been included in VB3, featuring the same feeding—from-live data mechanism present in VB2. An important improvement over VB2: now changes performed through SPARQL updates can be tracked and, consequently, put under validation if the project enables it and/or stored in the history.

4.9. Alignment

VB3 provides the same inter-project alignment support of VB2, allowing users to browse other projects and supporting semi-automatic label-based searches over them to provide candidate resources for alignments. In addition to this on-the-fly generation of mappings, VB3 introduces an alignment-validation tool: it is possible to load alignments following the model of the INRIA Alignment API [29], inspect the aligned resources and validate the alignment. Validated alignments can then be projected over standard RDFS/OWL or SKOS mapping properties, depending on the validated relation and the involved entities. E.g., two classes mapped through an inria:EquivRelation will be mapped through the property owl:equivalentClass while two SKOS concepts will be proposed to be aligned with a skos:exactMatch or a skos:closeMatch.

4.10. Declarative Service Implementation

An innovation that can be immediately appreciated by developers is the declarative way in which services are implemented: the business logic of the services is represented through a dedicated set of Java annotations that have been specifically developed for the VB framework. Declarations specifying if a service requires read/write access to the data, the capabilities the user must have in order to use them, the pre-requisites on the input parameters (e.g., if it represents an RDF resource declared in the dataset), etc. are all be represented in terms of this annotation vocabulary. Therefore, service development becomes an easier task, less prone to errors and the produced code is more readable, as the developer needs only to focus on what the service does.
4.11. Versioned datasets and metadata

In VB3, users can create snapshots of a repository (req. R12) and tag them with a version identifier (and other metadata, such as the time of creation of the snapshot). Users can travel across the different points in time identified by these versions, and thus analyze the evolution of browsed resources. The time-travel can be performed both globally, by switching version so that everything in the UI refers to the selected version, and locally, by inspecting different versions of a resource in the resource-view (Figure 7) or different versions of a tree (of classes, concepts, etc...)

4.12. Metadata Export

The “metrics” section of VB2 has been replaced with a page for editing and exporting metadata (R13) modeled after several existing metadata vocabularies: the Data Catalog Vocabulary (DCAT) [30], the Asset Description Metadata Schema (ADMS) [31], The Vocabulary of Interlinked Datasets (VoID) [32] and the Linguistic Metadata vocabulary (LIME) [33] (a lexical extension to VoID). While DCAT and ADMS mostly deal with static metadata, VoID and LIME offer statistical information about the dataset and its lexical information. The information of VoID and LIME is being computed through a profiler bundled with the LIME API [34]. This metadata build&export functionality is implemented as an extension point of the platform, so that new vocabularies can be dynamically added to the platform. For instance, an application profile for DCAT thought for European public sector data portals (DCAT-AP: https://joinup.ec.europa.eu/node/145996) has later been added to the list of exporters, as of the ISA² context specifically supporting public administration. Conceptual and lexical metadata will become important in a planned future version of the system, as this information will be exploited to support the setup of automatic alignment processes, in the spirit of [35, 36, 37].

4.13. Integrity Constraint Validation

A section dedicated to Integrity Constraint Validation (ICV) allows the user to inspect possible anomalies. These include violations of formal constraints (e.g. thesauri constraints on existence and uniqueness of preferred labels, disjointness between taxonomy and relatedness etc…) or problematic (though not necessarily illegal) patterns (e.g. a skos:Concept having a broader concept and being the top concept of a same scheme). Interactive fixes are provided (req. R3) for each discovered integrity break.

As of the forthcoming version 4.0 of VB3, a dedicated extension point for collaboration environments has been added to the platform. Collaboration environments are supposed to provide, in the context of defined projects, means to create and share tasks/issues/stories/discussions (shortly, collaboration items, or c.i.) that can be assigned to users for their resolution. This enhances coverage of requirement R2 for a controlled collaboration. The interaction with VocBench consists in functionalities for:

- creating projects into these collaboration environments (or identifying existing ones), associating them with VocBench projects
- creating collaboration items (or identifying existing ones) directly from RDF resources, in order to associate the c.i. with the RDF resource
- inspecting the full range of collaboration items pending resolution in a project
- inspecting the full range of collaboration items pending resolution on a given RDF resource
- being always informed, when inspecting an RDF resource through the resource-view, if there’s at least a c.i. associated to it and more specifically if there’s at least a pending one.
- Being directed to the page of the c.i. from the links to them available in VocBench

Currently, an implementation of the extension point has been realized for JIRA\(^\text{15}\) (Figure 8), the popular Project Management platform by Atlassian.

4.15. Desktop Tool and Collaborative Web Platform

As of requirement R7, the system offers a very lightweight installation (i.e. unzip and click-to-run) which, followed by default configuration options for both system and project creation, makes VB3 a good choice for users looking for a simple and easy-to-use desktop tool. Other more complex settings are still possible, satisfying different needs for distributed installation.

\(^{15}\) https://www.atlassian.com/software/jira
5. Impact

A heterogeneous user community has grown in these years around VocBench, including large organizations, companies needing VocBench as users and companies featuring it as their platform of choice in their range of offered RDF services, consultants needing a modeling environment etc...

Some of these long-lasting users still adopt VB2, but there are compelling reasons they will migrate to VB3: i) VB2 is no longer supported in terms of updates ii) the feature set of VB3 subsumes the one of VB2, iii) some features of VB2 have been substantially improved when migrated to VB3 (e.g. search, history/validation, etc...). Additionally, we should mention that VB3 has already relevant users. The Publications Office of the EU (which is managing the development of VB3) has been already using VB3 in production for their EuroVoc thesaurus and has recently adopted VB3 for collaboratively managing the Common Metadata Model ontology, thus exploiting the widened support of VB3 for OWL ontologies. FAO, which was the steward of VB2, is migrating to VB3 for the maintenance of their thesaurus Agrovoc in August 2018 and will soon adopt it for classification systems used in statistics. INRA, the French "Institut national de la recherche agronomique" (http://www.inra.fr/) and CIRAD, “la recherche agronomique pour le développement” (https://www.cirad.fr/) manifested their intention to move to VB3 (from VB2 and as a first adoption in respectively) for the management of their thesauri. Still in the field of agriculture, the US National Agricultural Library (NAL: https://www.nal.usda.gov/), together with FAO and CABI (“Centre for Agriculture and Bioscience International”, https://www.cabi.org/) agreed to use the VB3 platform in the context of the GACS project, a global agriculture thesaurus born from the integration of the three respective thesauri (NALT, Agrovoc and the CAB thesaurus) of these major players in the area.

The Senate of the Italian Republic is also migrating management of its thesaurus Teseo in these weeks from VB2 to VB3. There are also newcomers who are already adopting the platform and started directly with version 3, such as GelbeSeiten, the German Yellow Pages (https://www.gelbeseiten.de/), which are using VB3 to maintain their homonymous thesaurus, and the Solidaridad Network (https://www.solidaridad-network.org/).

Furthermore, VB3 acquired much more visibility and potential for adoption with respect to its predecessors, since it became the reference platform of the EU, recommended by the European Commission to the member states for the collaborative management of thesauri, ontologies and (now) lexicons. Only those with provable facts or a clear migration plan/intention. It does not include other relevant organizations which are evaluating the system and the many occasional users whom we are not in contact with

\[\text{http://agrisemantics.org/gacs/}\]

To improve our understanding of the uptake of VocBench, we collected some statistics from the download pages of VB2\(^{20}\) and VB3\(^{21}\) as of 1\(^{st}\) of July 2018. Figure 9 draws how many times each release of VB2 (on the left), respectively, of VB3 (on the right) was downloaded.

The most downloaded version of VB2 is the 2.3 with 392 downloads. The lower statistics associated with subsequent releases can be explained by the fact they were released one after the other with only a few months in between. These more frequent releases allowed us to deliver new features and (important) fixes to users; however, not every user had the same incentives to keep update with this faster release cadence. The last release of VB2 2.4.4 (available since 8 Match 2017) registered 279 downloads.

On the download page of VB2, there are also the documentations of VB 2.3 and VB 2.1 in PDF format, which have in total 910 downloads. A related observation is that the two versions of a simple thesaurus used in the getting started documentation were downloaded 420 times. Perhaps more important is the fact that a patch for a problem with the validation in VB 2.3 was downloaded 149 times: this figures tells us about users who considered the information they were validating valuable enough to warrant the application of the patch.

On the right side of Figure 9, we report analogous statistics for VB3. The first stable release (version 1.0) has 134 downloads, while the subsequent version 2.0 was downloaded 408 times (more than the most downloaded version of VB2). Version 3.0.1 (released five days after 3.0.0) has 345 downloads, still higher than the last release of VB2. These high figures are associated with relatively frequent releases (every few months), and thus support two (non-mutually exclusive) hypotheses: existing users are willing to test/use new versions as they bring substantial new features, or there is a sufficiently large supply of new users. The typical slowness of large organizations (which are typical adopters of VocBench) in reacting to changes (confirmed by the feedback we have gathered, where most of them declared to have “intention to move” or “being in the process of migrating” though still with an unclear schedule) suggests that the second hypothesis covers at least a non-trivial part of the phenomenon.

6. Conclusion and Future Work

In the last years, VocBench has addressed the needs of large organizations, companies and independent users needing an open source collaborative environment for editing thesauri, supporting a formalized editorial workflow. Continuous user feedback allowed us to spot bugs and to improve the usability of VocBench.

It thanks to this community feedback, to the support of the ISA\(^2\) program and to our desire to reach new quality levels that we started this endeavor, by rethinking most of VocBench from scratch, still benefiting from the experiences we had with VB2.

The most important achievement of the new platform lies at its core: a fully-fledged RDF core framework, developed by further improving the Semantic Turkey framework with functionalities for user management, role-based access control, change tracking and collaboration and by providing it with a new user interface. With respect to VB2, we could say VB3 is a mere user interface (delivered as a web application) for the improved Semantic Turkey platform. A second major achievement is the broadened support for all major standards of the RDF family, embracing OWL ontologies and, coming in July 2018, OntoLex (both as a core model for developing lexicons and as a lexicalization model for all kind of RDF datasets) which makes of VB a unique, comprehensive offer in the scenario of RDF development platforms. We hope that this evolution of the system will lay a solid foundation for the realization of a new range of services spacing from knowledge acquisition, evolution and management in the European and worldwide scenario.

Acknowledgments

This work has been funded by the European Commission ISA\(^2\) programme; the development of VocBench 3 (VB3) is managed by the Publications Office of the EU under contract 10632 (Infeurope S.A.)

References


---

\(^{20}\) https://bitbucket.org/art-uniroma2/vocbench2/downloads/ 

\(^{21}\) https://bitbucket.org/art-uniroma2/vocbench3/downloads/


