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

Editor: …
Solicited reviews: …

Armando Stellato[a], Manuel Fiorelli[a], Andrea Turbati[a], Tiziano Lorenzetti[a], Willem van Gemert[b], Denis Dechandon[b], Christine Laaboudi-Spoiden[b], Anikó Gerencsér[b], Anne Waniart[b], Eugeniu Costetchi[b], Johannes Keizer[c]
[a] *University of Rome, Tor Vergata, Via del Politecnico 1, 00133 Rome, Italy*
[b] *Publications Office of the European Union, Information Management Directorate, Standardisation Unit, Metadata Sector, 2985 Luxembourg, LUXEMBOURG*
[c] *GODAN secretariat, c/o CABI Head Office, Nosworthy Way, Wallingford, Oxfordshire, OX10 8DE, UK*

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 (AIMS) of the Food and Agriculture Organization of the United Nations[1] (FAO) 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 thesaurus development.

Rebranded as VocBench (VB), to suggest a more general environment for thesaurus management, the platform was later strongly reengineered in the context of a collaboration between FAO and the ART[2] group of the University of Rome Tor Vergata. The result of this collaboration, VocBench 2 (VB2) [2], released in 2013, was 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 and publication workflow. Under the hood, the original VB1 backend for RDF was replaced with the RDF Management framework Semantic Turkey (ST) [6, 7], already developed by the ART Group.

The strengths of the platform included the possibility for project administrators to define roles with very

[1] http://www.fao.org/

[2] http://art.uniroma2.it

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. They were appreciated especially by those organizations requiring a collaborative yet centrally controlled publication environment. Unfortunately, this controlled approach was realized on a rigid model comprising a few first-class resources (e.g. concepts, concept schemes, etc.) and predefined operations on them. Consequently, some other users felt as missing from the system more freedom on data modeling, desiring unrestricted capabilities for editing data at its very core, as in triple-oriented RDF editing environments. The increased 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.

VocBench 3 (or, simply, VB3) 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 ISA[2] Programme of the European Commission for "Interoperability solutions for public administrations, businesses and citizens"[3]. The action is managed by the Publications Office of the European Union[4]. 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 was released to the public on September 2017, under a BSD 3-clause license[5]. Since then, a second development iteration was carried on and terminated on July 2018 with VB3 v.4.0[6] (while v.2.0 and v.3.0 were released in the course of the iteration). The VocBench site[7] 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 third development iteration is being carried on, started after the fourth release of VB3 and terminating on June 2018.

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. This article is a revised and expanded version of a previous work [8], in which we previewed VocBench 3 just before the completion of its first development iteration. Improvements over that work include:

- discussion of related systems,
- discussion of additional features (e.g. extended input/output, integration of collaboration platforms, etc.),
- a more thoroughly analysis of the (meanwhile improved) support for OntoLex-Lemon,
- an example about XKOS showing the ability to support extensions to the core SKOS model,
- description of the system architecture,
- impact assessment.

2. Requirements

In this section, we list the foundational stones upon which VocBench 3 was developed, in terms of the requirements to be met. The 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). The original requirements have been reassessed and reformulated, accounting for the widened scope of the platform and its improved editing capabilities, while new requirements (R8-R15) have been added to complete the target objectives for the platform. The full set of revisions and new requirements has been collected following proposals by the VocBench developing team and requests performed by stakeholders through several means: dedicated stakeholder meetings held at the Publications Office, ISA[2] programme, pro-active feedback on the support mailing lists of the platform. The input from these different sources all contributed to lay

[3] https://ec.europa.eu/isa2/

[4] https://publications.europa.eu/

[5] https://opensource.org/licenses/BSD-3-Clause

[6] 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 is indeed a different project from VB2), so that the first release has been marked as version 1.0 of VB3.

[7] http://vocbench.uniroma2.it/

down the program for the development of VocBench 3, revised by the Publications Office and finally validated and approved by the ISA[2] committee.

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 progressively gaining momentum, thanks to data publication initiatives such as the Linguistic Linked Open Data[8] (LLOD) and to models for Ontology-Lexicon interfaces, such as *lemon* (lexicon model for ontologies) [9] and its most recent specification OntoLex-Lemon [10], realized in the context of the eponymous W3C community group[9].

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 a 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 Integrity. 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 one. 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. The system should support extensions (sometimes called plugins), which can provide additional capabilities, often bound to predefined extension points: for example, enable to export data to a new type of destination (e.g. an FTP server), or introduce a further serialization format (e.g. MADS[10]) .

R5. Data Scalability. The system must deal with (relatively) large amount of data, while 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 user interface 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. Support for OntoLex-Lemon lexicons and for ontology-lexicon interfaces was introduced later as a further requirement [11].

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

[8] http://linguistic-lod.org/

[9] https://www.w3.org/community/ontolex/

[10] http://www.loc.gov/standards/mads/

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 operations for conveniently creating/modifying complex descriptions of resources according to predefined modeling design patterns, 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 [12]. 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 User Interface. User interfaces 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 and project management information as well as history and 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 VB2, as a bird's eye view over the main layers of VB3 immediately reveals a more streamlined organization than the one of its predecessor. Indeed, as a result of the quick merge between FAO's collaborative platform VocBench and ART's Semantic Turkey (henceforth, ST), VB2'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 VB3, the web application (completely redeveloped using Angular[11]) is now a user front end 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 VB2 (and in ST in the specific) is the strict adoption of the RDF4J[12] framework. Semantic Turkey's RDF API used in VB2 were based on OWLART[13], an abstraction layer supporting access – through dedicated implementations of the abstraction – to different RDF middlewares, such as Jena[14] [13] and Sesame [14]. 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

[11] https://angular.io/
[12] http://rdf4j.org/
[13] http://art.uniroma2.it/owlart/
[14] http://jena.apache.org/

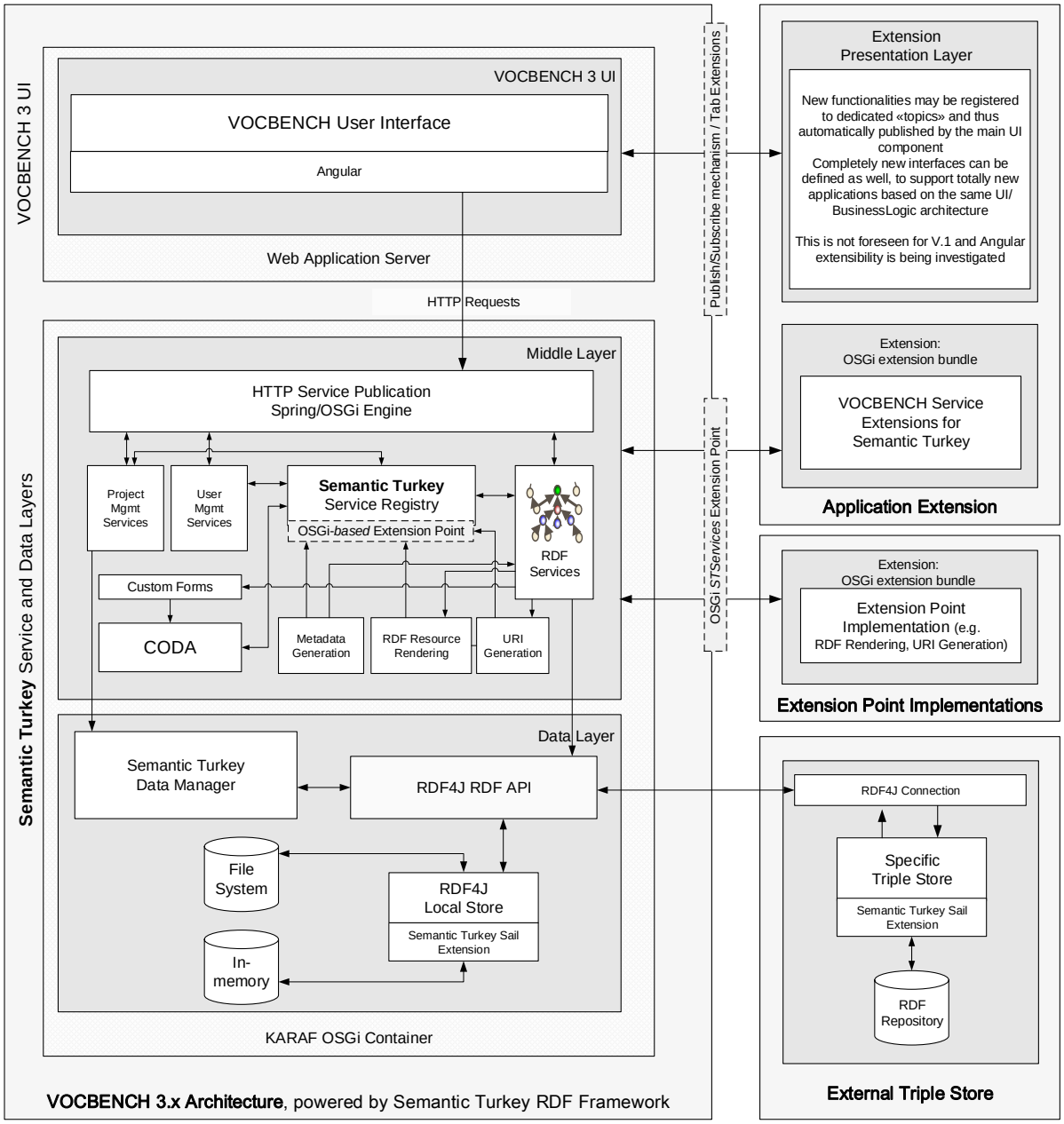

Figure 1. VocBench 3 Architecture

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 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 know a-priori).

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 stance and adopt that RDF framework as the official RDF API for ST. This choice relieved us from developing our own middleware, and moreover offered us a plethora of framework-specific solutions that we could exploit. Notably, we embraced the RDF4J's sail-mechanism (Storage and Inference Layer) for developing storage-side components (represented under the local and remote triple stores in the architecture view) for intercepting effectively-modified triples (see section 3.2 and section 4.4).

3.2. Extendible Architecture

The adoption of the OSGi (formerly Open Service Gateway initiative)[15] service framework allows for dynamic plugging of extensions (see requirements R4 and R9). 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 realize a pluggable implementation of an extension point by creating:

- An implementation of the extension interface associated with the functionality (this will also automatically associate the extension with 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 VB.

While there is currently no native extension mechanism for web interfaces built through Angular technology, the user interface of VB 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, in order to make extensible the user-visible features of VB3 (see Section 4)

- *CollaborationBackend*: support for different collaboration platforms (see Section 4.15). Currently an instance for JIRA[16] is available.
- *DatasetMetadataExporter*: support for the representation and export of metadata about the edited dataset, according to different vocabularies (see Section 4.13)
- *Input/Output related extension points* (see Section 4.8):
  - *Deployer*: a deployer allows VB to deploy data to some destination type. 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

[15] https://www.osgi.org/

[16] https://www.atlassian.com/software/jira

  - *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*: these components can be used in data processing chains such as export and import procedures. Transformers can be used to filter out unnecessary/unwanted information or to create new data, such as publication metadata or, for instance, redundant data such as materialization of inferred triples. The general applicability of RDF transformers made them a reusable 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 [15]. 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 VB. Each rendering engine provides a SPARQL fragment that is usually connected to the queries of the various resource retrieving services. See Section 4.1.3 for a discussion of this extension point in the context of supporting multiple lexicalization models.
- *RepositoryImplConfigurer*: this consists in configuration templates for different kinds of triple stores. Currently, implementations are available for the various storage solutions of RDF4J and for Ontotext GraphDB[17] (formerly OWLIM [16]). These implementations support the dyanamic generation of configuration dialogs (see Figure 2), which allow users to customize the repository being created using options that are specific to the chosen storage solution. In the example in Figure 2, the user chose an RDF4J Native Store, and then the configuration dialog presented specific options: whether to force the syncronization of the filesystem to the non-volatile storage device (a performance vs durability tradeoff), which indexes to use (depending on the query patterns), and which types of reasoning perform (which, obviously, are not costless).
- *SearchStrategy*: a search component that can provide different strategies for the search of resources. The appropriate strategy depends on the storage solution chosen for a repository, in particular, because different triple stores offer different mechanisms for text search. Indeed, SPARQL only supports text searches via a FILTER with some string matching functions. However, as FILTERs are often applied to discard solutions rather than find them, their use as a search mechanism is quite inefficient if the number of candidates is very high (e.g. tens of thousands). To solve this problem, we provided an implementation of the *SearchStrategy* that can be used together with GraphDB to take advantage of its non-standard full-text search capabilities.
- *URIGenerator*: a pluggable component for the automatic generation of URIs. Indeed, it is possible to devise a large number of strategies to mint new URIs, depending on diverse technical and political arguments. Ontologies often use human-friendly but language-specific identifiers based on a label (e.g. `http://schema.org/Person`), in order to make data more easily consumable (without specific tool support). Conversely, thesauri and other multilingual datasets frequently adopt alphanumeric, language independent, identifiers (e.g. `http://aims.fao.org/aos/agrovoc/c_2993`). While the *URIGenerator* packed with VB is highly configurable, it is possible to develop specific generators that comply with custom patterns that cannot be generated by it.

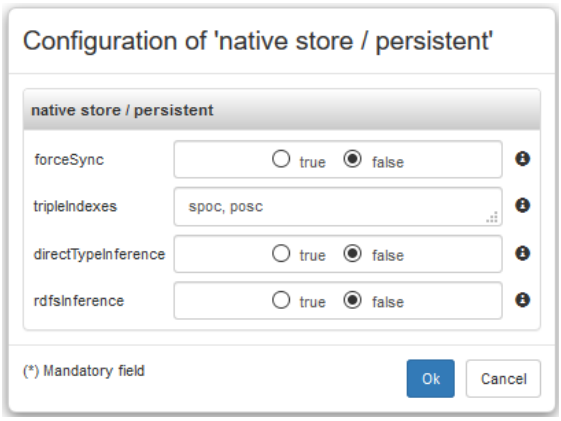

Figure 2. Dynamic dialog for the configuration of a repository based on some store solution (in this case, the RDF4J Native Store)

[17] `http://graphdb.ontotext.com/`

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 now 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[18] [17] 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[19] [18] and GATE[20] [19]. Therefore, CODA extends UIMA with facilities and a powerful language – PEARL[21] [20] - 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 forms* (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[22] [21], 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 reengineered for the new version, improved and enriched with a new user interface specifically developed for VocBench 3.

4. VocBench 3 New and Improved Features

In this section we list 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 3) is the element that vividly marks the difference between VB3 and its predecessors. The *data* view, letting the user explore the project's dataset, is the one that mostly represents this difference. The several tabs of VB2 (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 and 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. This capability is necessary to support the development of RDF data in general (requirement R8), marking a profound departure from previous versions of the system, which had dedicated panels for certain types of resources that

[18] http://art.uniroma2.it/coda/
[19] https://uima.apache.org/
[20] https://gate.ac.uk/
[21] http://art.uniroma2.it/coda/documentation/pearl.jsf
[22] http://art.uniroma2.it/sheet2rdf/

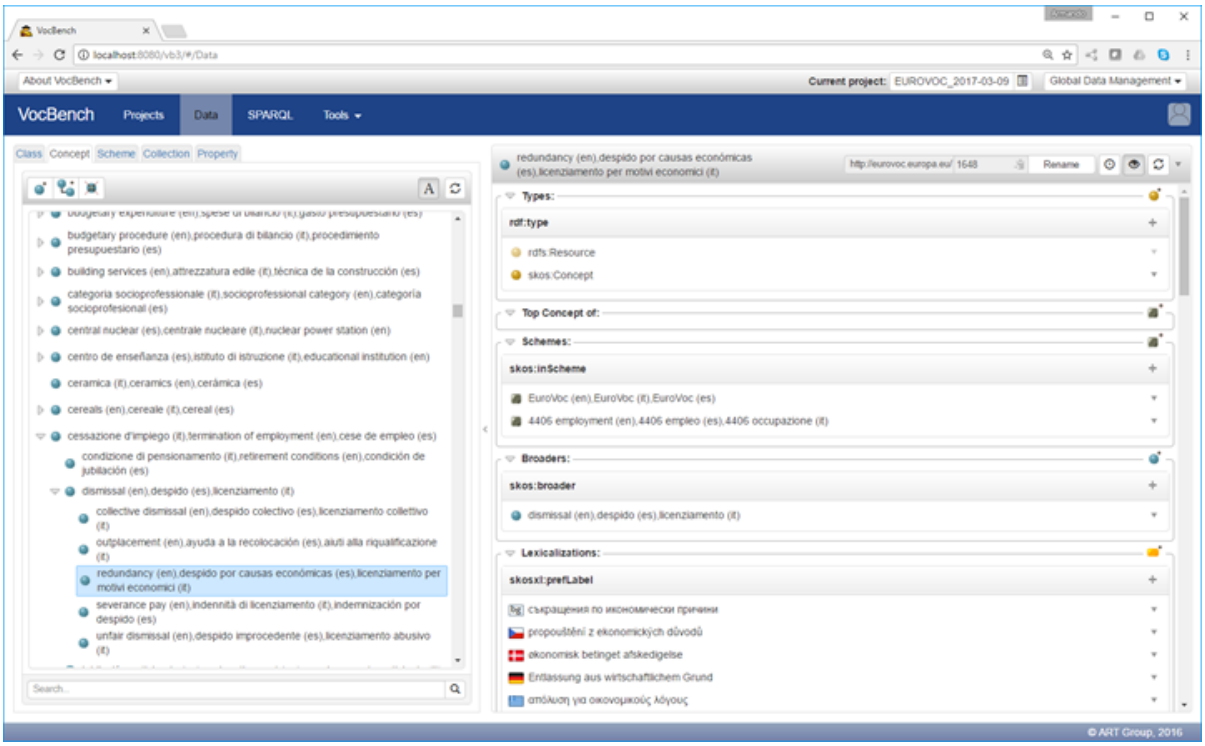

Figure 3. VocBench UI showing EuroVoc (http://eurovoc.europa.eu)

occur in thesauri (e.g. concepts and reified labels). It is worthy of note that the general applicability of the resource-view to any resource and the possibility to edit any of their details from it also concur to satisfy requirement R10.

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 rdf: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. The XKOS [22] vocabulary, for instance, defines some subproperties of the SKOS semantic relations (i.e. skos:broader, skos:narrower and skos:related), differentiating between generic and partitive hierarchal relations or between diverse associative relations. Figure 4 shows the resource-view of the concept "steering wheel": its *broaders* section tells not only that "car" is broader than "steering wheel", but also that the former is an holonym of the latter (because of the use of the property xkos:isPartOf). Indeed, a subproperty of skos:broader can be indicated (optionally) both in the dialog for the creation of a new concept (when creating it as narrower of an existing concept, thus refining the relation among them) and in the dialog for the addition of a broader concept. The previous example shows the ability of the user interface to handle extensions (e.g. XKOS) of the core modeling vocabularies (e.g. SKOS), concurring to fulfill requirement R8.

4.1.2. Dataset Model and Lexicalization Model

A special mention goes to the *lexicalizations* section: it represents an abstraction over different kinds 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

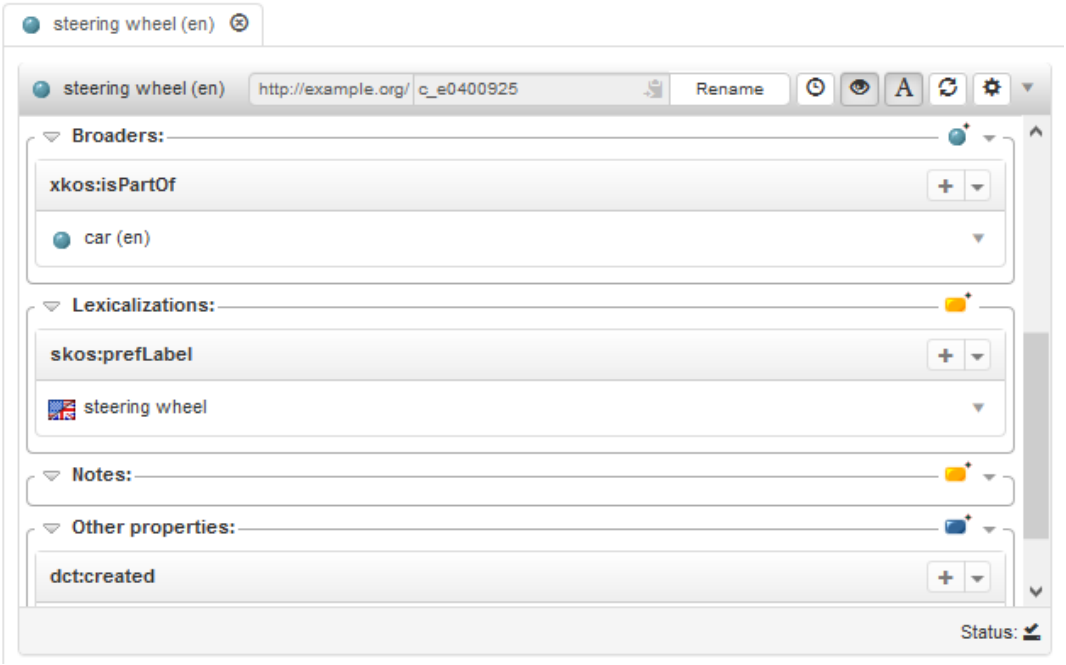

Figure 4. Visualization of specialized hierarchical relationships between SKOS concepts

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 with 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 indirection 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 3) 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 three tabs with a class tree and instance list, a property tree and a datatype list respectively, 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 5). Modern thesauri are (usually) organized around a hierarchy of concepts that can be easily

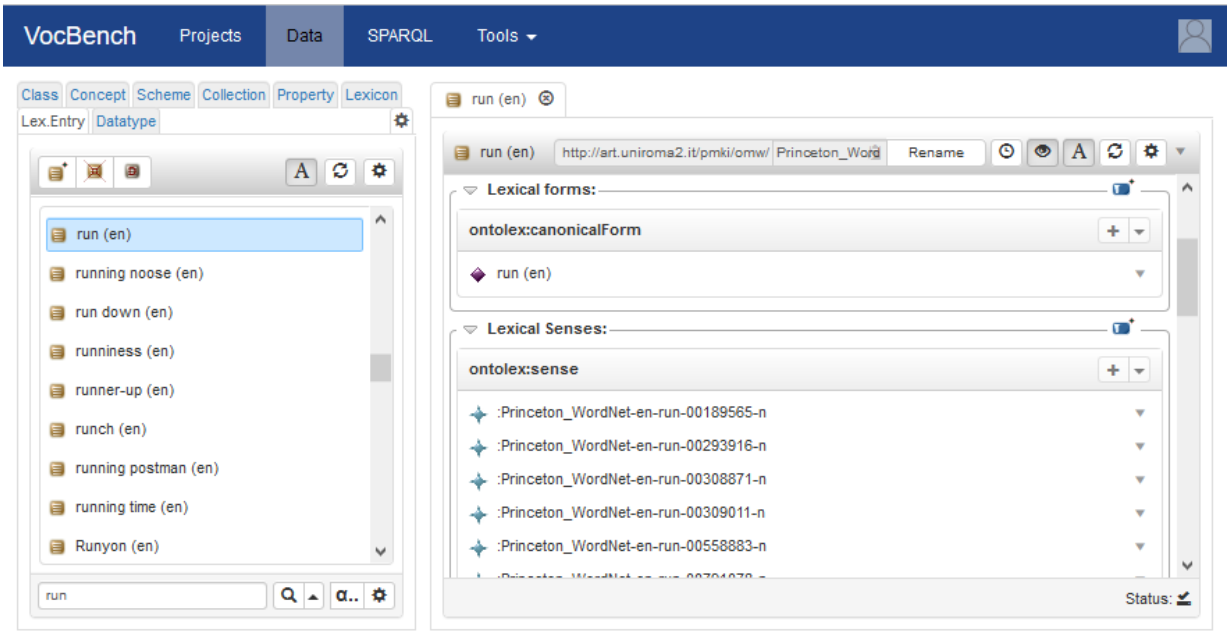

Figure 5. Visualization of WordNet's Lexical Entries in OntoLex views. In fact, the system is managing the whole collection of 34 wordnets collected by Open Multilingual Wordnet [64]

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[23] [23], corresponds to the notion of *synset*, the semantic index that glues together sets of synonyms (hence, the term synset), 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 properties, 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 and 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[24] (qualified name) 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 engine* extension point (see Section 3.2). The default renderer behaves in a way similar to VB1 and VB2, 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 VB, 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

[23] `https://wordnet.princeton.edu`

[24] A qualified name (qname) is formed by a prefix, followed by a colon, and then a local name (e.g. `schema:Person`). If the prefix occurring in a qname is associated with a namespace URI (e.g. `http://schema.org/`), then the qname can be understood as an abbreviation for the URI obtained concatenating the namespace URI and the local name (e.g. `http://schema.org/Person`)

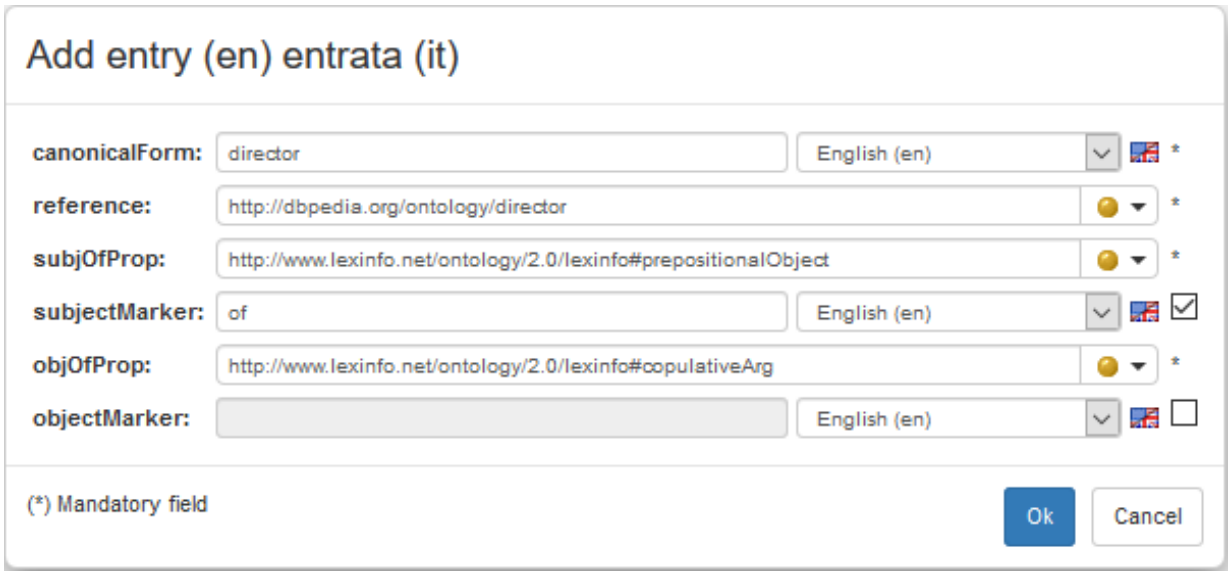

Figure 6. Custom Form for a relational noun in the OntoLex-Lemon model

forms rely on the combination of the following four key elements:

- 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 6) based on the above declarations, prompting for information that is required for "constructing" a new resource

Custom forms have been described more in details in [24], which analyzed and evaluated their expressive power by applying them to the use case of representing lexical entries using the OntoLex-Lemon vocabulary. In that work, a subset of the *lemon* Design Pattern Library [25] was implemented as custom forms: VB exploits them to generate a form-based interface for the creation as well as the visualization of diverse lexical entries. Figure 6 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 y and the stereotypical verbalization *y is the director of x*.

4.1.5. Administration Pages

Another relevant difference in the UI 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 VB 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 prerequisites 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 VB can handle multiple projects, which can also be interlinked for mutual data access (e.g. for purpose of alignment). VB 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 (or, equivalently, RETRIEVE), 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. We recall that "CRUD is [...] the most common acronym for the four basic functions of persistent storage: create, retrieve, update and delete" [26]. Our language supports all these functions, which are identified by their initial letter. Actually, we extended CRUD with a fifth letter 'V' standing for VALIDATE: this permission grants the right to validate other users' work (see Section 4.5).

The language is implemented as a series of facts for the Prolog [27] 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[25] [28] engine, developed in Java by the University of Bologna. In the web client, jsprolog[26], 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 7). 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), *thesaurus 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 Section 4.5)

[25] http://apice.unibo.it/xwiki/bin/view/Tuprolog/WebHome

[26] https://github.com/Sleepyowl/jsprolog/

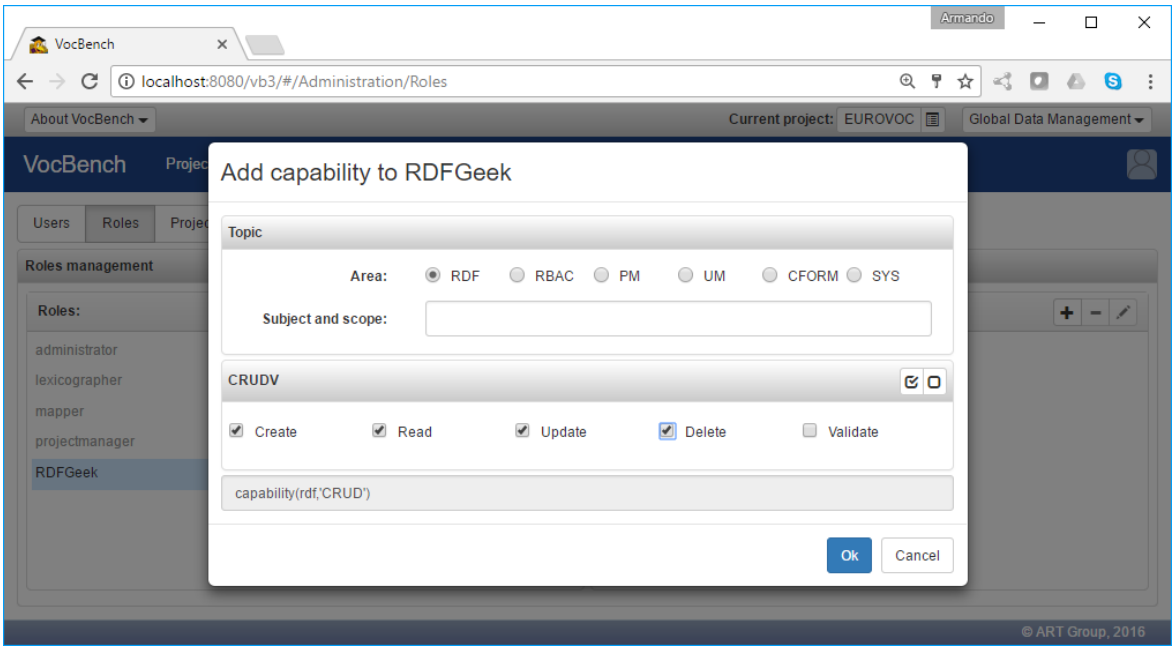

Figure 7. Editing a capability for a role in VocBench

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* and *validation* was appreciated by many users. However, being based on a predefined 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 state of the dataset (i.e. a history reflecting all changes performed, which would enable the recreation of previous states of the repository). Furthermore, the separate history system (inherited from the original VB 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 (req. 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 (see Section 3.1). 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 Ontotext GraphDB.

The design of the history and change tracking mechanism in VB3 was guided by a landscape analysis [29, 30] in which we discussed the nature and the representation of change, reviewed relevant 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-step 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.

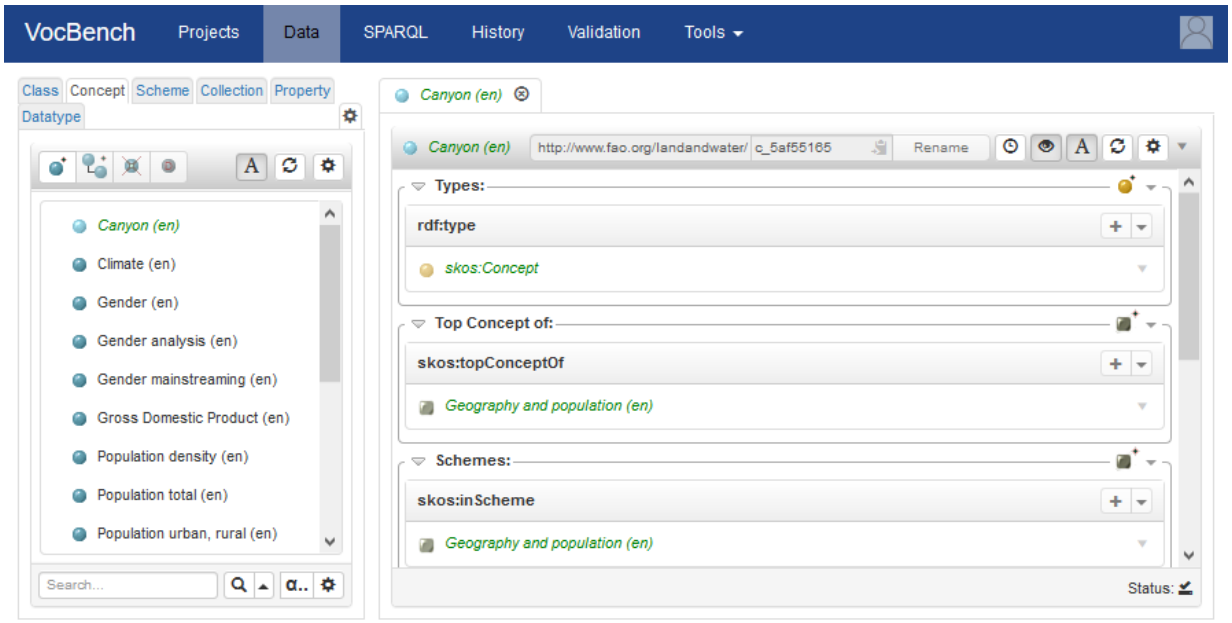

Figure 8. Proposed concept "Canyon" shown in the concept tree, as well as in the resource-view

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. We refer the interested reader to [31] for an in-depth discussion of different reification mechanisms, and an analysis of their performance impact. Some mechanisms deeply affect how data is represented (e.g. singleton properties [32], or even standard reification), while named graphs [33] would be sufficiently lightweight, if VB did not already use them to differentiate between local and imported triples. Nonetheless, as discussed below, VB3 eventually adopted named graphs for the representation of proposed triples.

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* (i.e. the graph where managed information is stored) 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 double validation as useless. Probably, the original intention, dating back to the first VB, 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 VB. 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".

In Figure 8, we show the concept "Canyon" that was created in a project with validation. The triples describing this concept are asserted in the *staging-add-graph* (instead of the project *main graph*): the resource-view shows them with a combination of green color and italic font, while triples in the *main graph* (none in this example) are displayed in black with a normal font. Since the assertion of the characterizing type (skos:Concept) is also staged for addition, the concept itself is considered proposed, and thus displayed in the concept tree differently from other (already validated) concepts. Figure 9 shows the creation of the concept in the list of operations pending for validation: a user with suitable rights (see Section 4.3) can decide to either accept or reject these operations. Accepting an operation makes its effects permanent (i.e. updating the *main graph*), while rejecting an operation undoes its effects (i.e. remove the previewed triples in

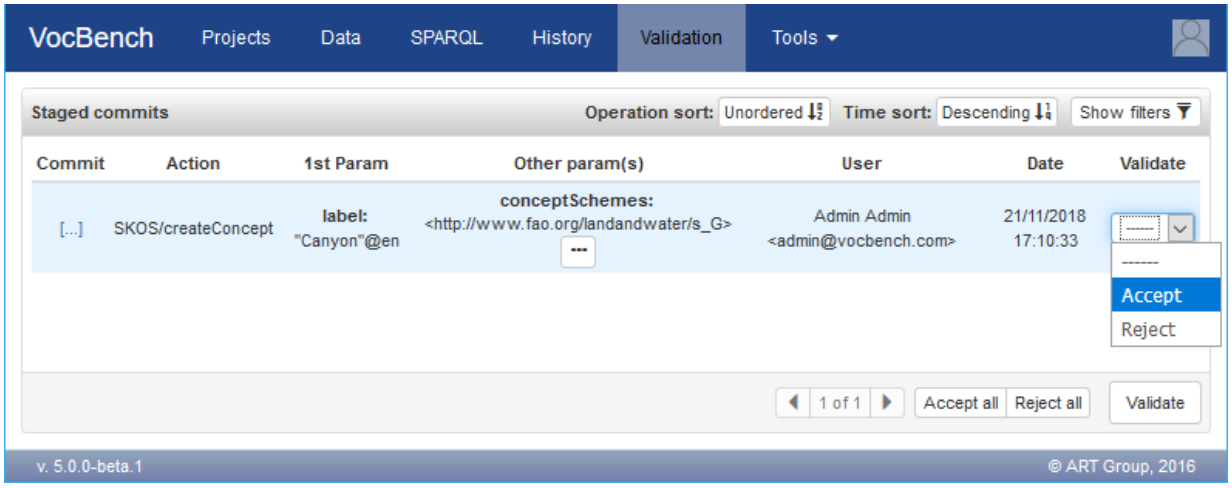

Figure 9. Accepting the creation of a concept to move it from "proposed" to "validated"

the *staging* graphs). In the latter case, no trace of the operation remains, nor is the operation logged in the history: rejecting an operation should be like the operation was never performed. Future directions for the system foresee the possibility to store (in a logically-separated space) rejected actions so that, should they be re-proposed in the future, the user can be informed and discouraged from submitting them for validation again.

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

VB already allowed for importing ontology vocabularies for modeling thesauri. Now VB also supports ontology development (requirement R8), with editing of OWL axioms (using the Manchester syntax for both editing and visualization, see Figure 10) and an almost full coverage of OWL2 expressions. VocBench delegates reasoning to the triple store that manages the ontology and its data (see Section 3). Nonetheless, VB is aware of the difference between explicit and inferred statements: they are rendered differently in the user interface, and users may decide whether inferred statements should be included in the resource-view of some resource (see Figure 10). Currently, VB does not present justifications of the inferences, because RDF4J (see Section 3.1) does not provide any mechanism to obtain them from the underlying triple store. The kind of inference support provided by triple stores (with respect to dedicated inference engines) is in fact generally more purposed towards efficient real-time materialization over large data rather than providing end-user features such as justifications.

Triple stores usually have a rule-based implementation of reasoning, often associated with the total materialization strategy: the entailment closure of the ontology is computed in advance by the iterative application of the rules defining the semantics of the ontology modeling language. By referring to the triple stores commonly used with VB, we highlight how actual implementations of this reasoning strategy differ in flexibility, expressiveness and optimization (with respect to different workloads).

The reasoner shipped with RDF4J only supports the RDFS ontology language: the corresponding rules are hard-coded into the code of the reasoner. Moreover, when knowledge is retracted from the ontology, this reasoner computes the entailment closure from scratch, since it cannot determine which entailments are no longer supported. This approach is clearly inefficient, when the workload includes many deletions that do not affect the entailment closure significantly, e.g. the deletion of a factual assertion is usually less far reaching than the deletion of a schema-level assertion. An alternative "schema caching" reasoner is not rule-based, since it gathers schema-level assertions and creates a

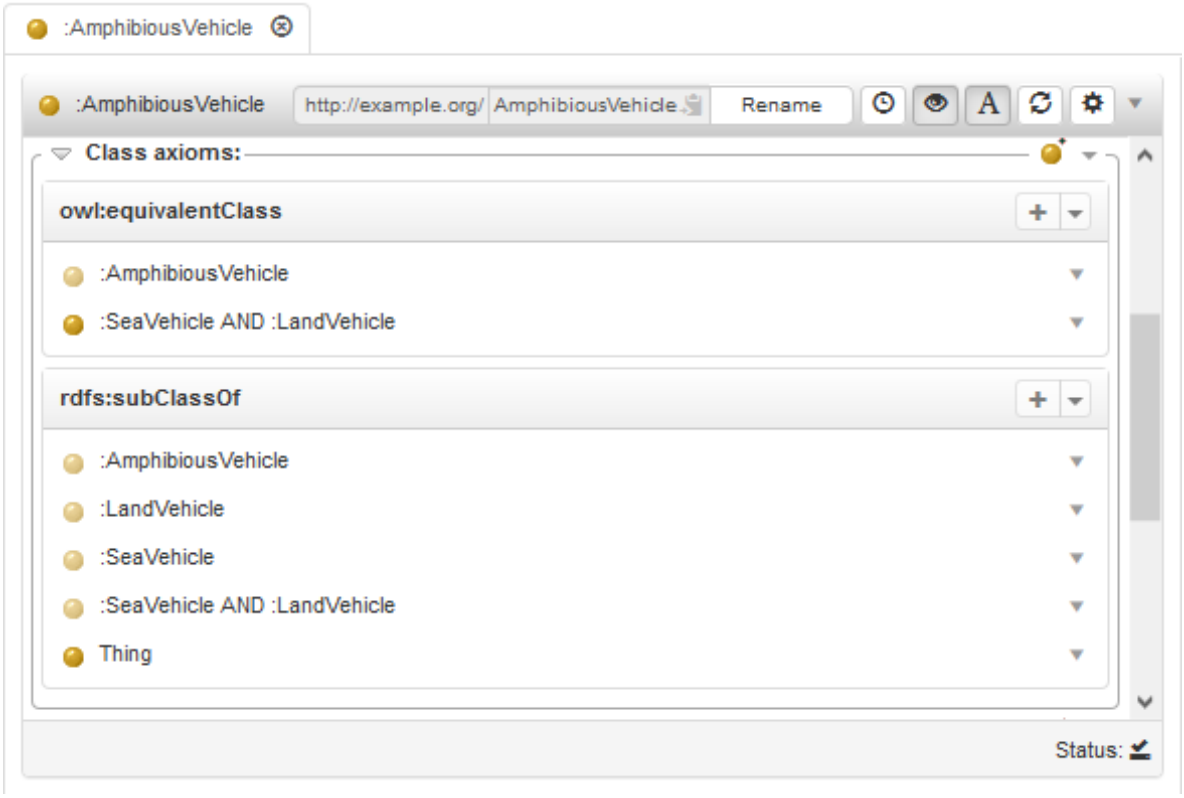

Figure 10. Resource-view of the class `AmphibiousVehicle`, showing the use of the Manchester syntax to visualize class descriptions and the different rendering of inferred statements (lighter color). The button with the eye icon controls the inclusion of inferred statements

data structure to quickly determine inferred statements. This reasoner better handles deletions that do not affect the schema, while being more performant (up to 80x faster) and more complete (but still bound to the RDFS semantics). For these reasons, the former forward-chaning resoner is going to be deprecated in favor of this new one in the forthcoming RDF4J 2.5.

Ontotext GraphDB implements reasoning using a rule-engine that can execute arbitrary rules written in a proprietary rule-language. GraphDB provides by default optimized rulesets for RDFS, OWL2-RL and OWL2-QL semantics. When statements are retracted, GraphDB first determines (by forward-chaining) their logical consequences, and then removes only those inferred statements for which it is not possible to determine (by backward-chaining) any derivation not including the knowledge being retracted. GraphDB can also perform consistency checking and prevent modifications that would produce inconsistent knowledge. GraphDB features a non-rule implementation of `owl:sameAs` that eliminates the need to store N^2 identity links and to rewrite statements for each equivalence class.

4.8. Input/Output

VB3 features completely redesigned data load and export capabilities. Since they are somehow specular, we will focus hereafter on the export procedure, using the example of Figure 11 dealing with export of a SKOS-XL thesaurus.

The data inside a project is stored in multiple graphs (the *main graph*, zero or more *graphs for imported ontologies* and zero or more *staging graphs*, if there are modifications pending for validation). By default, VB3 exports only the *main graph* (containing the data edited by the users) but it is possible to choose other graphs. In our example, we accept the default and only export the *main graph*.

In some circumstances, the triples in the chosen graphs can't be exported as they are: some information shall be added or removed, data shall be modelled differently, or only a subset of the data is required. VB2 partially satisfied this need in the narrowed context of thesauri, by allowing to export only a given concept (optionally, with its children) or a concept scheme as

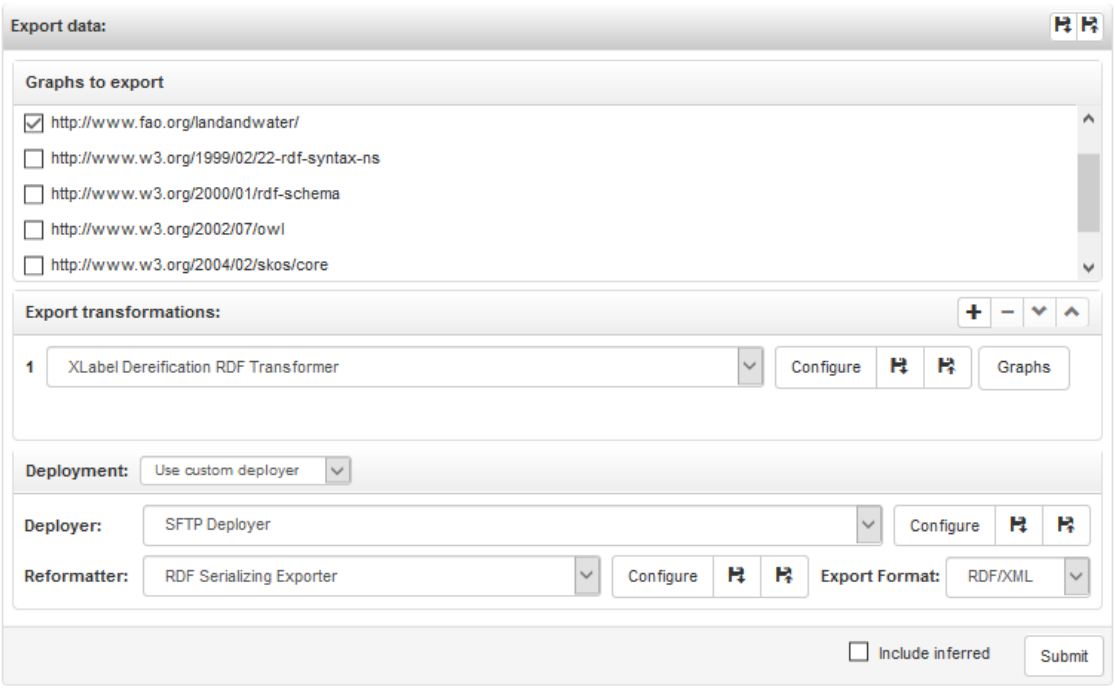

Figure 11. Export data

a whole. The wider scope of VB3 clearly required a more general solution, in the form of *rdf transformers* (see the corresponding extension point in Section 3.2). In VB3, the data being exported can be processed by a chain of these components, each performing a specific transformation. Since *transformers* operate destructively, they are executed on a temporary copy of the data. In our example, we select the *transformer* that turns SKOS-XL reified labels into plain SKOS labels.

The RDF data being exported, optionally processed by a chain of transformers, should go somewhere. Currently, VB3 supports three destinations: a *file*, a *triple store* or a *custom destination*. Both *files* and *custom destinations* require that the RDF data is serialized into a byte sequence conforming to some data serialization format. To this end, the user can configure an *RDF reformatting exporter* (see the corresponding extension point in Section 3.2). In the example, we simply adopted the *RDF Serializing Exporter* (featuring all standard RDF serialization formats) and serialized the data into the RDF/XML syntax, but a *reformatting exporter* for the Zthes [27] format is provided as well. While VB3 implements the export to file natively, the *triple store* and *custom destinations* are implemented via the extension point *deployer* (see Section 3.2), which implements a specific communication protocol. In the example, we chose to deploy the RDF/XML serialization of the transformed data to an SFTP server.

It is worthy of note that these complex export chains can be saved and reused (even across projects). In particular, even the configurations of each of the individual components (*transformers*, *reformatters* and *deployers*) can be saved and reused in different export chains. In our example, this means that the SFTP destination can be configured once (with host, port, username, password, etc.) and reused in other export chains.

4.9. SPARQL Querying and Update

A new SPARQL UI, based on YASGUI [34] 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*.

VB3 also introduced the possibility to store queries (with different visibility scopes, following the classification of component scopes described in section 3.2)

[27] http://zthes.z3950.org/

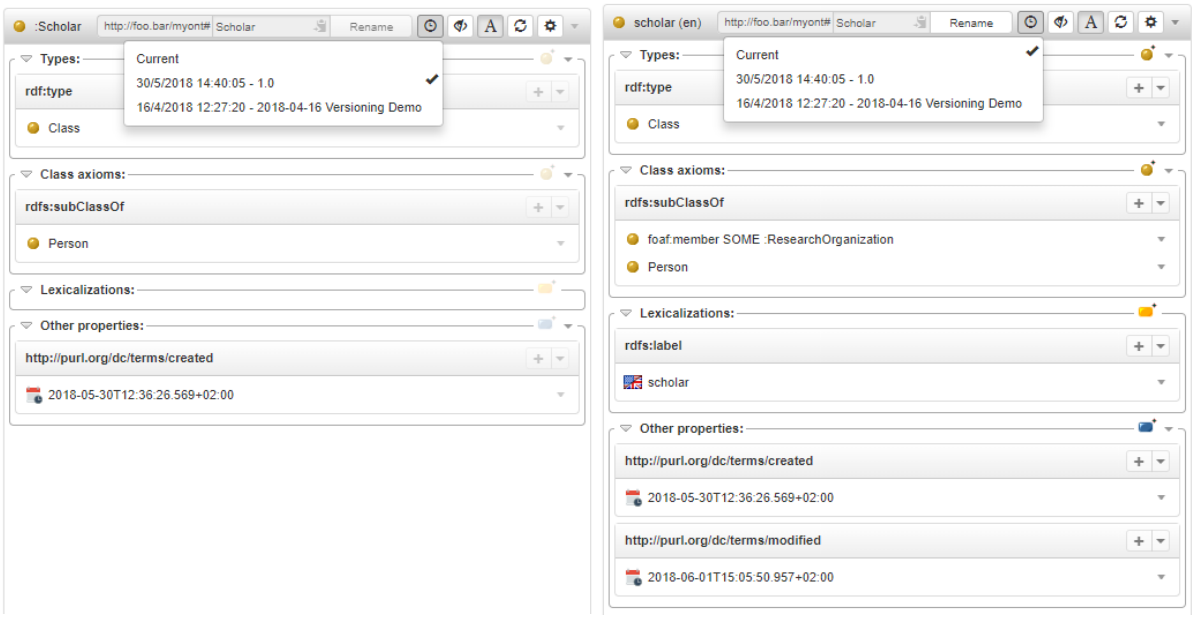

Figure 12. time-traveling across different versions of a resource

for later reuse, and for sharing them with other users, even across projects. This capability is useful in different cases: i) a query is sufficiently complex that it is convenient to retrieve and reuse it, ii) a user writes a query that can be used by other users less proficient (or no proficient at all) with SPARQL, iii) exactly the same query shall be executed periodically (e.g. for analytical tasks).

VB3 offers several options for downloading the results of a query, including various tabular formats for tuple queries, and RDF serializations for graph queries. Regarding the latter case, we observed that it is not much different from exporting the RDF data in a project (see Section 4.8): following this intuition, we supported a customizable chain of *transformers* as we did in the export.

4.10. 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 [35], 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.

4.11. 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 prerequisites on the input parameters (e.g. if it represents an RDF resource declared in the dataset), etc. can 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.

4.12. 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

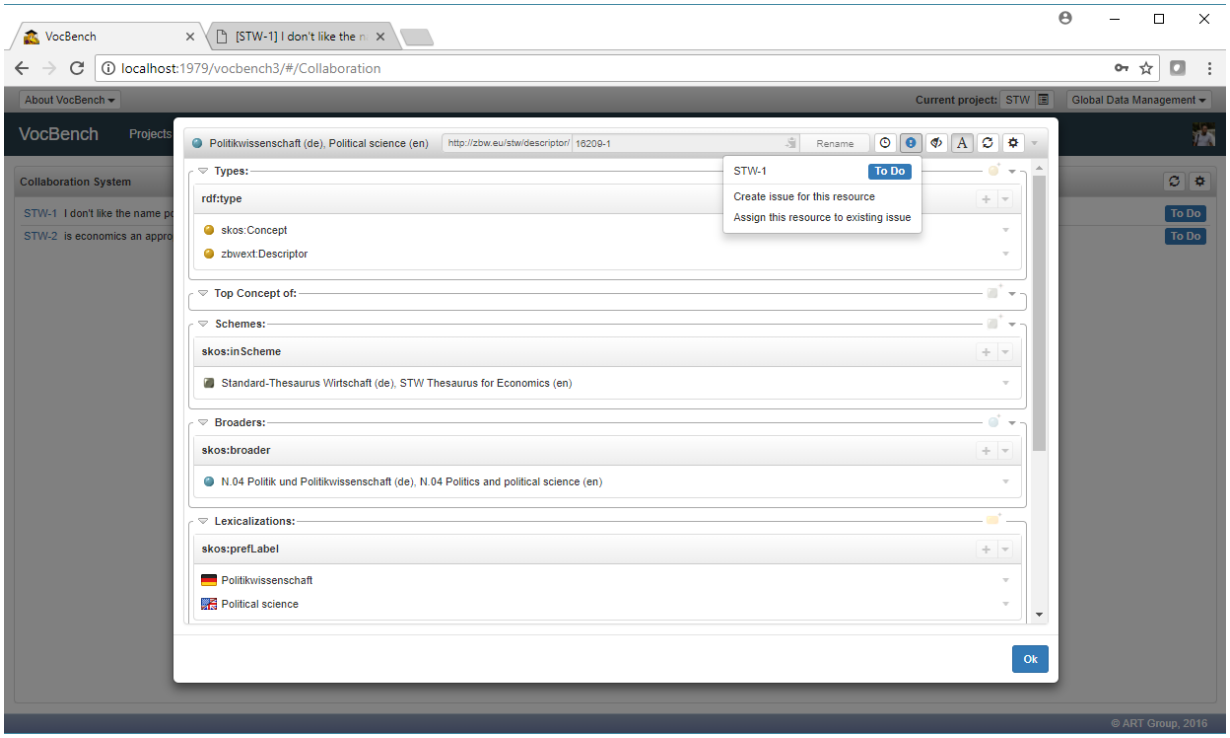

Figure 13. Listing issues in JIRA associated to concepts in a thesaurus, opening one of these concepts and checking other collaboration options

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 12) or different versions of a tree (of classes, concepts, etc.)

4.13. 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) [36], the Asset Description Metadata Schema (ADMS) [37], The Vocabulary of Interlinked Datasets (VoID) [38] and the Linguistic Metadata vocabulary (LIME) [39] (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 [40]. This metadata build and export functionality is implemented as an extension point of the platform, so that new vocabularies can be dynamically added to the platform (see extension point *Dataset Metadata Exporter* in Section 3.2). For instance, an application profile for DCAT thought for European public sector data portals (DCAT-AP[28]) has later been added to the list of exporters, as of the ISA[2] 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 [41, 42, 43].

4.14. 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

28 https://joinup.ec.europa.eu/node/145996

broader concept and being the top concept of a same scheme). Interactive fixes are provided (req. R3) for each discovered integrity break.

4.15. Collaboration Environments

A dedicated extension point for collaboration environments has been added to the platform (see extension point *collaboration backend* in Section 3.2). Collaboration environments are supposed to provide, in the context of defined projects, means to create and share tasks/issues/stories/discussions that can be assigned to users for their resolution. This enhances coverage of requirement R2 for a controlled collaboration. The interaction with VB consists in functionalities for:

- creating projects into these collaboration environments (or identifying existing ones), associating them with VB projects
- creating collaboration items (or identifying existing ones) directly from RDF resources, in order to associate the collaboration items 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 collaboration item associated to it and more specifically if there's at least a pending one.
- being directed to the page of the collaboration items from the links to them available in VocBench

Currently, an implementation of the extension point has been realized for JIRA[29] (Figure 13), the popular Project Management platform by Atlassian.

4.16. 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 (separation of data servers, UI servers), better performance, etc.

5. Impact

A heterogeneous user community[30] has grown in these years around VB, including large organizations, companies needing VB 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 for which 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[31]. The Publications Office of the EU (which is managing the development of VB3) has already adopted VB3 in production for their EuroVoc thesaurus and for collaboratively managing the Common Metadata Model [44] ontology, thus exploiting the widened support of VB3 for OWL ontologies. The Publications Office is also, since this October 2018, providing hosting and support to Directorate Generals (DGs) of the European Commission for managing their datasets. At the time of writing (only 4 months after the decision to provide this support) the production installation of VocBench 3 at the Publications Office is hosting 47 projects for 10 DGs, with each DG and the Office itself having more than one team working on different projects. The number of adopters exploiting this supported hosting within the European Commission and related institutions is rapidly growing.

FAO, which was the steward of VB2, migrated to VB3 for the maintenance of their thesaurus Agrovoc in August 2018 and has recently started adopting it for classification systems used in statistics. INRA, the French "Institut national de la recherche agronomique" [32] and CIRAD, "la recherche

[29] https://www.atlassian.com/software/jira

[30] http://vocbench.uniroma2.it/support/community.jsf

[31] The list expressed in the following paragraph has been gathered through a mix of direct interaction with known partners/users and data harvested from a questionnaire advertised on the VB mailing list. In both case, explicit consent has been given to the authors for disclosing the information they have reported. This list is obviously limited to our most close interactions and, among those, only those with provable facts or a clear migration plan/expressed 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

[32] http://www.inra.fr/

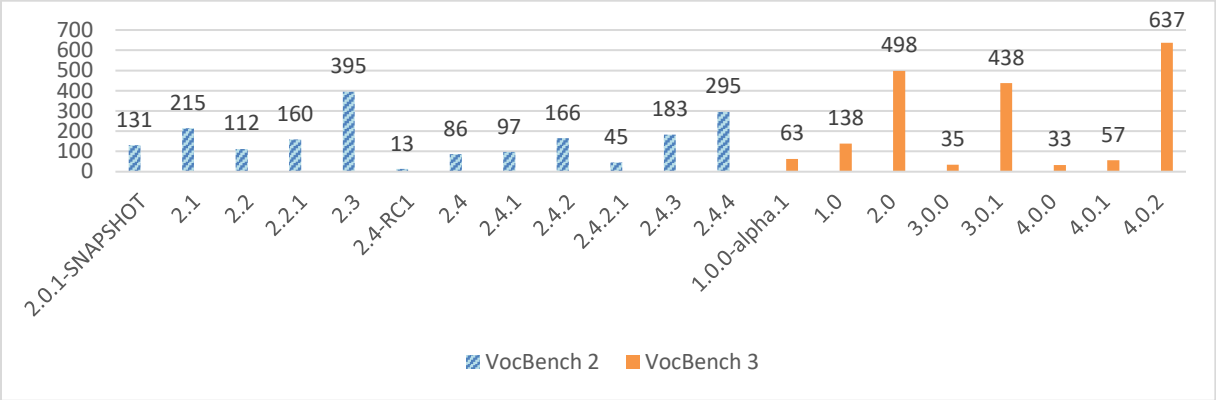

Figure 14. Download statistics for VocBench 2 (on the left) and VocBench 3 (on the right) as of 8 February 2019

agronomique pour le développement"[33] 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)[34], together with FAO and CABI ("Centre for Agriculture and Bioscience International")[35] agreed to use the VB3 platform in the context of the GACS[36] 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 has also migrating management of its thesaurus Teseo from VB2 to VB3. There are also newcomers who are already adopting the platform and started directly with version 3, such as GelbeSeiten[37], the German Yellow Pages, which are using VB3 to maintain their homonymous thesaurus, and the Solidaridad Network[38].

Furthermore, VB3 acquired much more visibility and potential for adoption with respect to its predecessors, since it became the reference platform of the EU[39], recommended by the European Commission to the member states for the collaborative management of thesauri, ontologies and (now) lexicons.

To improve our understanding of the uptake of VocBench, we collected some statistics from the download pages of VB2[40] and VB3[41] as of 8th of February 2019. Figure 9 draws how many times each release of VB2 (on the left), respectively, of VB3 (on the right) was downloaded.

The most downaloded version of VB2 is the 2.3 with 395 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 candence. The last release of VB2 2.4.4 (available since 8 March 2017) registered 295 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 959 downloads. A related obseravation is that the two versions of a simple thesaurus used in the getting started documentation were downloaded 491 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 14, we report analogous statistics for VB3. The first stable release (version 1.0) has 138 downloads, while the subsequent version 2.0 was downloaded 498 times (more than the most downloaded version of VB2). Version 3.0.1 (released five days after 3.0.0) has 438 downloads, while version 4.0.2 (released three days after 4.0.0) has 637 downloads being the most downloaded so far. These high figures are associated with relatively frequent releases

[33] https://www.cirad.fr/
[34] https://www.nal.usda.gov/
[35] https://www.cabi.org/
[36] http://agrisemantics.org/gacs/
[37] https://www.gelbeseiten.de/
[38] https://www.solidaridadnetwork.org/
[39] https://ec.europa.eu/isa2/solutions/vocbench3_en

[40] https://bitbucket.org/art-uniroma2/vocbench2/downloads/
[41] https://bitbucket.org/art-uniroma2/vocbench3/downloads/

(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.

The user community's main point of interaction and support is the VocBench discussion group [42]. The group (as of 8th February 2019) counts 154 members. In order to roughly quantify the impact of the group, we compared its size with the analogous community forum of RDF4J[43]. The user group of RDF4J counts 390 members, so the sizes of the two groups are in the same order of magnitude. Additionally, we have to consider that RDF4J is one of the two most popular Java frameworks for RDF (the other one is Jena), thus relying on a less scattered distribution of users than RDF editing platforms may have.

User questionnaires are occasionally sent to the mailing list of VocBench in order to gather feedback from users about satisfaction and ideas for new directions for the system. A recent investigation held in March, 2018, based on a questionnaire diffused on the revealed that roughly 50% of the respondents already adopted VocBench 3 in production, 22% were in the process of adopting it, 17% were already using VocBench 2 in production and were considering moving to VocBench 3 while 11% was shared by users adopting VB2 and still not sure whether to move to the new platform and newcomers evaluating the VB3 platform. Adoption of VocBench 3 is still mainly focused on thesaurus development: among those adopting VB3 in production, 89% of the respondents are using it for developing thesauri while 33% is adopting it for OWL ontology development. We do not have figures for OntoLex development as at the time of the investigation the OntoLex editor in VB had not yet been released.

6. Related Work

VB3 supports a variety of core RDF knowledge models with dedicated facilities, and it can be used both as a desktop tool and as a web-based collaborative environment. The number of systems it could be compared against is consequently very large. Instead of attempting an exhaustive survey of these systems, we focused our attention on selected systems, which are currently and widely recognized as references in their area of specialization.

Protégé[44] [45] is an open source ontology development system, which is extremely popular. Since the inception of its web-based incarnation, we should distinguish more carefully between the traditional desktop system and the newcomer WebProtégé [46]. Nonetheless, in the rest of this section we will use the unqualified Protégé to refer to the desktop system.

Protégé was originally based on a *frame* language with a subsequent extension for OWL. Protégé 4 was redesigned to use the OWL API [47], and it became complaint with OWL-DL.

Protégé benefits from DL reasoners. HermiT [48] is a notable example: fully compliant with the OWL2 Direct Semantics, it offers a range of services including class/property classification and (beyond OWL) DL-safe rules, while being high performing and scalable to large resources.

DL semantics precluded most meta-modelling, which is seamlessly supported by RDF systems (such as VB3), based on OWL Full semantics and equipped with rule-based reasoners. OWL2 closed this gap with the introduction of punning. Hogan *et al.* summarized the relative strengths of each approach, noting [49] that tableau-based (DL) reasoners are usually less scalable and efficient than lightweight rule-based reasoners, which, however, are less expressive.

One of the strengths of Protégé lies in its extensible architecture and accompanying plugin framework. At the time of writing, the plugin library[45] of Protégé contains 120 items. One of such plugins is SKOSed [50], which consists in "a suite of views and tools for working specifically with SKOS". SKOSed adds convenience (e.g. the hierarchical representation of SKOS concepts) on top of the foundation laid down by Protégé, which guarantees the possibility to intermix SKOS and OWL (e.g. to extend SKOS with additional classes or properties), and to leverage already existing features (e.g. reasoners). Under this viewpoint, SKOSed is very close to how VB3 achieves its multi-model support, even though SKOSed offers a notably smaller set of functionalities with respect to VocBench

[42] http://groups.google.com/group/vocbench-user
[43] http://groups.google.com/group/rdf4j-users
[44] http://protege.stanford.edu/
[45] https://protegewiki.stanford.edu/wiki/Protege_Plugin_Library

and is limited to the desktop version of Protégé. Moreover, SKOSed is no longer maintained, and the last version is declared compatible with Protégé 4.1+: a quick check suggests that SKOSed is not compatible with the more recent Protégé 5.2.

In addition to the single-user local usage, Protégé desktop supports a client-server mode, otherwise known as multiuser model. Different instances of Protégé (clients) connect to a shared, central instance (server), so that different users can work together on the same ontology (or SKOS dataset, in case of SKOSed). The communication between the clients and the server is based on the Java-specific RMI protocol, while VB3 uses a lightweight protocol based on HTTP and JSON, as commonly found in Web APIs.

The implementation of the client-server mode in Protégé 3 is very different from the one found in Protégé 4. The main difference is that in Protégé 3 changes are sent individually to the server as they are performed in the clients, while in Protégé 4 changes are grouped together and only sent when the user decides to commit them[46]. In Protégé 3, the clients are thus tightly bound to the shared server, and they cannot work without a connection to some server. This scenario is conceptually similar to a web application (in particular a single-page application, such as VB3), although clients are locally installed rather than downloaded on the fly from the web. Conversely, in Protégé 4, the clients can work offline, as the connection to the server is only required when performing server-related operations, such as commit or update. In fact, the ontology can even be changed outside of Protégé: upon commit, Protégé will determine the changes by computing the difference between the current version and the one originally fetched from the server. The client-server mode of Protégé 4 is thus conceptually close to a centralized version control system specialized for ontologies.

Indeed, version control systems that are widely popular in software development (e.g. GIT) are often considered inadequate for versioning of ontologies and RDF datasets. Firstly, they can only version a serialization of ontologies or datasets in some (line-oriented) concrete syntax. Secondly, but perhaps more importantly, they can be easily confused by semantically irrelevant operations, such as rearrangement of statements or change of blank node local identifiers.

Nonetheless, VoCol [51, 52] demonstrated the viability of GIT[47] for distributed development of vocabularies, by proposing a methodology and a suite of integrated tools. VoCol orchestrates the execution of different tasks (e.g. validation, documentation generation, etc.) in response to notifications sent from the GIT repository, often in response to events such as pushing of new commits. VoCol is not an ontology editor on its own. Actually, the wiki of VoCol suggests to edit the files versioned with GIT using a text editor, since some ontology/RDF editors tend to completely change the serialization of the data (e.g. the order of the triples) upon saving the updated version[48].

Collaborative Protégé [53] is an extension for Protégé 3 that introduces a number of collaboration-oriented features beyond the multiuser model (client-server mode). Noteworthy features include the annotation of ontology components and changes, the management of a change history, inline discussions and a chat. In VB3, history is managed internally, while other capabilities are delegated to external (already appreciated) services (e.g. JIRA for issue management), through extensible connectors, while at the same time offering a smooth integration (e.g. showing the issues associated with a resource inside its resource-view).

In addition to maintain an history of the changes applied to an ontology, WebProtégé enable users to download the ontology in each state of its evolution. This is a rather unique feature, since other systems (VB3 included) only supports discrete snapshots of the edited data. Actually, a similar behavior may be found in VoCol, since in GIT it is possible to download any (previous) state of a repository.

WebProtégé was initially thought as a lightweight (open source) ontology editor, but it has grown over time reducing the functional gap with respect to the desktop version. Indeed, it supports complex class expressions expressed in the Manchester syntax (as VB3), and even the editing of SWRL rules (not found in VB3).

WebProtégé uses Collaborative Protégé, which provides support for change tracking, notes, discussions and access control; however, in WebProtégé, users can also monitor ("watch") individual resources or entire branches of the class tree: changes to these resources are listed in the user interface, or sent via email notifications. This feature may increase the pace of the exchanges between editors co-working on the same area

[46] https://protegewiki.stanford.edu/wiki/Protege4ClientServer

[47] https://git-scm.com/

[48] https://github.com/vocol/vocol/wiki/Developing-Vocabularies-with-VoCol-Environment

of the ontology. Unfortunately, this kind of notifications is not already supported by VB3.

WebProtégé supports the acquisition of instance data by means of customizable forms that effectively shield domain experts from the complexity of the underlying ontological modelling. This mechanism was borrowed from Protégé 3, and it is somewhat related to VB3 custom forms. Actually, VB3 custom forms are meant to enhance the construction of an instance or the setting of a property value, and they do not aim at replacing the editing panel (i.e. the resource-view) as a whole.

Currently, WebProtégé does not support plugins like its desktop counterpart[49]; however, its modular architecture makes it easy to modify the code of the system and introduce additional components (e.g. custom tabs or custom form widgets). Conversely, VB3 already supports plugins at the service level, with the dynamic generation of configuration dialogs in the user interface. Nonetheless, provisioning of arbitrary extensions for the user interface (as available in Protégé desktop) is still missing in VB3, due to limitations of the Angular technology.

In line with its use in the life science domain, WebProtégé provides a dedicated facility for linking terms from BioPortal[50] [54] ontologies. Starting from version 5.0, VB3 is going support data catalogs in general, and provide interesting features based on them (e.g. importing an ontology from a catalog).

Currently, WebProtégé lacks dedicated facilities for SKOS, nor can third-party plugins like SKOSed fill this gap. Fortunately, there are numerous alternative SKOS editors, most of which – especially the most recent ones – are web-based collaborative editors. Mochón *et al.* surveyed [55] existing thesaurus managements tools from the perspective of LOD. They found that PoolParty[51] outperforms other systems with respect to every indicator, while TemaTres[52] [55] ranked second. In the following paragraphs, we will discuss these two systems, as well a few others (also included in the aforementioned survey), which we considered noteworthy either for impact or for diversity of approach.

PoolParty is a highly regarded suite of semantic technologies, including (among others):

- PoolParty Thesaurus Server (PPT) Basic/Advanced: thesaurus management
- PoolParty Extraction Server (PPX): term and phrase extraction, entity linking and disambiguation
- PoolParty Graph Search Server (PPS): semantic search and data mining

These technologies are proprietary, and sold in different combinations (as a one-time purchase or as a renewable subscription). For example, PoolParty Enterprise Server bundles PoolParty Thesaurus Server Advanced and PoolParty Extraction Server. There exists a demo server for this offering that can be accessed free of charge for 30 days after registration. This offering covers to a full extent the thesaurus (and ontology) management capabilities provided by the suite. Actually, the more comprehensive (and costly) PoolParty Semantic Integrator differs for its inclusion of semantic search & data mining capabilities and the possibility to use a wider spectrum of triple store backends. The latter is supported by VB3 by a dedicated extension point, with bundled implementations for some triple store technologies.

In PoolParty, the analogous of the resource-view for a concept consists of several tabs corresponding to various functionalities, such as detailed editing, notes, history, quality management, etc. The editing tab itself is subdivides into tabs, corresponding to different cohesive groups of properties. By default, it is possible to use the SKOS tab to edit the core description of a concept. However, additional tabs can be activated, by choosing an ontology or a custom scheme (a view on a subset of one or more ontologies developed for some purpose). Indeed, PoolParty treats ontologies as second-class citizens: while it has long been possible to edit them inside PoolParty, until version 7, editing of ontologies used a completely different, very constrained user interface. Indeed, only the recently published PoolParty 7 introduced a class tree for the class taxonomy.

PoolParty can track changes to a thesaurus, while it is possible to browse both the global history of a project and the history of individual resources. The approval workflow allows to exert some control on the proposed changes: concepts transitions to the draft state upon any change that affects them, and an explicit action is required to transition them again to the approved state. If the proposed changes are not approved, the concept can be assigned to a user for remedy. If the thesaurus is seriously compromised, it is possible to

[49] https://mailman.stanford.edu/pipermail/protege-user/2018-March/008741.html

[50] https://bioportal.bioontology.org/

[51] https://www.poolparty.biz/

[52] https://www.vocabularyserver.com/

revert to a previously snapshotted state. VB3 also supports snapshots and allows users to quickly switch to a snapshot for visualization. However, reverting to a snapshotted state is – at the time of writing – not supported. PoolParty supports quality management through the integration of qSKOS[53] [56], which can evaluate a number of quality criteria. These criteria can be enforced interactively (e.g. forbid the creation of a new concept with the same preferred label of another one) or only included in quality reports. Indeed, they inspired the development of our integrity constraint validation subsystem.

PoolParty can leverage existing Linked Datasets: for example, it is possible to populate a thesaurus or create mappings based on existing datasets (e.g. DBpedia). A similar aim can be found in VB3: the possibility to browse (via the resource-view) any resource that can be accessed via HTTP or via a registered SPARQL Endpoint, the possibility to create individual mappings to remote datasets, etc.

PoolParty supports (as VB3 does) the association of JIRA for task management.

While it is not, strictly speaking, a thesaurus management feature, interesting features of PoolParty (in some offerings) are the acquisition of concepts/terms from a corpus and the semantic annotation/classification of documents based on a thesaurus. Currently, VB3 lacks these capabilities. One may argue that these capabilities should be provided by external services that (right now) can interact with VB 3 through its API or via a shared triple store. Indeed, there have already been experiences integrating knowledge acquisition from text in our platform [57, 58], based on CODA and the analytics framework UIMA.

TopBraid EVN (Enterprise Vocabulary Net) is another proprietary web-based editor, which supports different types of asset: thesauri, ontologies, corpora, content tag set and crosswalk. Differently from PoolParty, TopBraid EVN supports ontologies to the same extent as thesauri. Moreover, the management of these diverse assets is in fact very uniform, at least for what concerns metadata management, access policy definition, import/export, etc. Actually, the main differences arise in the editing section, where unsurprisingly ontologies and thesauri use a different interface than, for example, cross-walks. In the following, when not specified otherwise, we will refer to the editing of thesauri and ontologies.

Governance of changes to an asset is implemented by TopBraid EVN through the notion of working copies. Working copies are alike branches in GIT, since they can be edited autonomously, and then merged to the production version of the asset, if the proposed changes are approved. The approval process is supported by showing the difference between the copy and the production version in terms of property value changes of resources. This type of comparison report is also available to editors, to understand how a resource has evolved, and in case undo a change. In addition to this resource-centric view, TopBraid EVN also supports browsing and searching the history in terms of coarse-grained changes consisting of multiples triple additions/removals. Like the case above, it is possible to revert individual changes. The documentation warns about reverting changes in the middle of the history, because that operation might create orphan resources. Indeed, this limitation also affects the validation machinery of VB3. The behavior of TopBraid EVN differs from the validation in VB3 with respect to their interaction with the history. Reverting a change appends to the history another change that undoes the effects of the former. In VB3, a change pending for validation is – on purpose – not logged in the history, and if it is rejected, it is like the change had never happened. Actually, this behavior can be achieved in TopBraid EVN via the mechanism of working copies. A working copy can also be archived to achieve something comparable to the snapshots supported by PoolParty and VB3.

TopBraid EVN implements integrity checks via SPIN[54] (SPARQL Inference Notation) and, more recently, via SHACL (Shapes Constraint Language) [59]. These constraints can be evaluated interactively or included in a batch report (like PoolParty does). The use of a standard, declarative language might be advantageous for supporting the addition of new constraints. In VB3, constraints are written in the code, and there is some duplication between the checks performed by the services and the integrity constraint validation subsystem: this low-level approach enables dedicated visualizations and different, contextualized quick fixes to ease debugging and repair of anomalies.

Similarly to WebProtégé, TopBraid EVN allows to customize the form associated with a class. While the form is represented internally through SWP [55] (SPARQL Web Pages), it can be created (to some extent) through a graphical editor inside the web interface.

[53] https://github.com/cmader/qSKOS/
[54] http://spinrdf.org/
[55] http://uispin.org/

TopBraid EVN supports the assignment of tasks to users at different levels, from asset-wide tasks to the level of individual resources in an asset. Additionally, TopBraid EVN supports comments on resources. As comments can transition between a few statuses, they can also be used as a simple task system.

In addition to these integrated facilities for collaboration, TopBraid EVN supports linking projects to the collaboration platform JIRA: e.g. from the editing view associated with a resource, it is possible to see or create related issues on JIRA. As already pointed out when discussing a similar capability found in PoolParty, VB3 also features a smooth integration with JIRA. Actually, VB3 aims at the possibility to plug in different collaboration platforms, by supplying an implementation of the proper extension point.

Like PoolParty, TopBraid EVN can manage document corpora, and supports both manual and automatic (document-level) semantic tagging of documents.

TemaTres is an open source web application for the management of thesauri, glossaries, controlled vocabularies, etc. As already said, it ranked second after PoolParty in the survey of Mochón *et al.* [55].

It uses a term-based model in contrast to the concept-based model underpinning SKOS. Indeed, SKOS is only available as an import/export format, while data is stored in a relational DB using a dedicated schema. Actually, this schema is enough flexible to support the definition of custom relations; however, it doesn't support the import of custom ontologies, and these customizations do not make it into the SKOS export anyway.

TemaTres supports multilingualism via interlinked monolingual vocabularies: these can be managed by the same TemaTres instance or accessed via a terminological web service. This web service interface is implemented by TemaTres itself, thus laying down the foundation for the federation of TemaTres instances. Beyond multilingual mapping, federation is also useful for general vocabulary mapping: it makes it possible to look up a remote vocabulary for a match of a term. In VB3, we achieve a similar capability through the "assisted search mapping", which allows to lookup a SPARQL endpoint for a resource matching a given search criterion. An interesting feature of TemaTres is the possibility to generate mappings in bulk, by downloading and then reversing mappings contained in a federated vocabulary.

TemaTres features a workflow model similar to the one of VB3: concepts are created in a candidate state, and once accepted, they do not reenter a "modified" state after each modification. While TemaTres implements the state as an explicit metadata of the term (as in VB2), in VB3 the state is represented implicitly through the state of the triple asserting the characterizing class of the resource (when validation is enabled).

TemaTres has a very limited support for change history, as it supports to browse recent changes to a vocabulary (also available as an RSS feed).

TemaTres can be configured to enforce some integrity constraints (e.g. disallow polyhierarchy), while offering some quality assurance tools (e.g. terms without hierarchical relationships).

The TemaTres Keyword Distiller uses the terminological web service implemented by TemaTres to keywords from a controlled vocabulary using terms extracted automatically from text[56].

iQvoc[57] is a vocabulary management system based on SKOS by innoQ Deutschland GmbH. Its development started within a research project exploring challenges and opportunities of Linked Data publication via state-of-the-art web frameworks. Specifically, iQvoc uses the open source framework Ruby on Rails. Despite an initial binding to a specific vocabulary, subsequent development focused on separating the core logic from vocabulary-specific customizations. When iQvoc 3.0 was open sourced, even the support to SKOS-XL was moved to a separate module[58]. iQvoc is similar to TemaTres under several viewpoints, such as resources being displayed as directly addressable content-pages, federation, bulk-generation of mappings, etc. On the other hand, iQvoc profoundly differentiates from TemaTres, being natively based on the SKOS model. However, iQvoc does not work at the level of RDF statements, but SKOS is implemented as a domain model persisted to a relational DB via the ORM (Object-Relational Mapping) facility found in Ruby on Rails. Extensions to the domain model have to be implemented in code by additional modules.

Governance of change in iQvoc is based on the idea of differentiating the published version of a concept and its (unique) draft version created upon a change to that concept. Furthermore, conflicts are avoided by locking the draft version of a concept for single usage. Users with suitable rights can publish the drafted version of a concept and make the changes persistent and

[56] https://vocabularyserver.com/wiki/index.php?title=Tematres_keywords_distiller

[57] http://iqvoc.net/

[58] https://github.com/innoq/iqvoc_skosxl

visible. Before publishing a draft version, these users can explicitly request the evaluation of some integrity constraints, in order to avoid the corruption of the data being edited. In iQvoc, the granularity of validation coincides with individual concepts being edited, while VB3 records individual operations for validation. On the one hand, the coarse grain approach of iQvoc allows to group together individual changes that are best vetted together. However, the scope of the validation is limited to individual concepts, and it is therefore impossible to validate cross-concept changes.

iQvoc does not support a global history of changes; however, changes to individual resources can be described by attaching SKOS change notes to the affected resource (while the system sets the creator and the date time automatically).

The aforementioned systems do not have a dedicated facility for editing OntoLex-Lemon lexicons. Actually, VB3 is – to our knowledge – the first multi-model editor concerned with OntoLex-Lemon. In literature we could only find purpose-built systems that are tailored to a specific application of *lemon* (in broad sense, thus including the legacy Monnet *lemon* [9], which influenced the development of OntoLex-Lemon). Montiel-Ponsoda *et al* argued [60] that generic data-driven editors are difficult to use with the *lemon* model: some model constructs are not properly displayed, and, moreover, there are constructs that are realized as a bunch of triples, although from a logical viewpoint they should be created and edited as a single entity. Montiel-Ponsoda *et al* addressed these issues through the development of Lemon Source. Beyond the aforementioned problems, Lemon Source addresses collaborative development of lexicons, which was identified as a key requirement by the authors. In doing so, the system had to incorporate change tracking, role-based access control, validation workflow, etc. Most of the items in this list are generally useful features, and already found in general-purpose collaborative editors such as VB3. Both Lemon Source and VB3 hide (most of) the complexity of the model, by prompting the user for essential information about a given construct (say the lemma of a lexical entry), so that its complex RDF serialization can be generated automatically. Actually, Lemon Source automates the construction of ontology lexicons a bit further: by means of a configurable NLP (Natural Language Processing) pipeline, it creates lexical entries from (RDFS) labels commonly found in ontologies, computing their tokenization, syntactic tree, and using (whenever possible) entries in already existing lexicons (for example, Princeton WordNet). Lemon Assistant [61] is another web-based editor for *lemon*, which is based on the design patterns for ontology lexicons [25]. Its motivation lies in the observation that the use of *lemon* is complex and error-prone. In VB3, we implemented all the patterns supported by Lemon Assistant, but we complemented them with more general operations on *lemon* models, so that we are not constrained to ontology lexicons. Indeed, despite what the extended name ("lexicon model for ontologies") of the model would suggest, *lemon* is largely used beyond its original scope, for example for the representation of language resources as Linked Data. However, VB3 does not support – at the moment – the generation of usages examples of lexical entries and some integrity (cross-language) checks.

LexO [62, 63] is the most recent *lemon* editor, and it is particularly focused on the needs of digital humanities (e.g. attestation). Still under development, LexO is only available as a web-accessible demo. The inspection of this demo revealed that the system is very tailored to the editing of lexicons with a very basic support for ontologies (e.g. hierarchical visualization of the class taxonomy).

In Monnet *lemon*, the representation of a wordnet was achieved as if it were a lexicalization of an ontology: synsets were usually represented as instances of an arbitrary class, while words were represented as lexical entries. Previous editors supported the editing of language resources with this mechanism. However, OntoLex-Lemon eventually differentiated between an ontology lexicon and a wordnet-like resource: the class *lexical concept* models synsets, while the attachment of lexical entries is achieved with a dedicated set of properties. Obviously, editors developed long before OntoLex-Lemon may not support all of this. Conversely, VB3 features an extensive support for lexical concepts, and it was tested with very large resources: for example, in Section 4.1.3 we cited the possibility to manage the collection of 34 wordnets collected in Open Multilingual Wordnet. Another recent innovation of the model not supported by previous editors is the LIME metadata module. VB3 supports the computation and the export of this kind of metadata. In the future, VB3 will benefit from this metadata to better orchestrate the alignment of different datasets: e.g. determine the overall in the supported natural languages, or the identification of potentially useful language resources.

7. 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 is 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 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, going beyond SKOS thesauri and embracing OWL ontologies and 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 ISA² programme; the development of VocBench 3 (VB3) is managed by the Publications Office of the EU under contract 10632 (Infeurope S.A.)

References

[1] C. Caracciolo, A. Stellato, A. Morshed, G. Johannsen, S. Rajbhandari, Y. Jaques and J. Keizer, "The AGROVOC Linked Dataset," *Semantic Web Journal*, vol. 4, no. 3, p. 341–348, 2013.

[2] A. Stellato, S. Rajbhandari, A. Turbati, M. Fiorelli, C. Caracciolo, T. Lorenzetti, J. Keizer and M. T. Pazienza, "VocBench: a Web Application for Collaborative Development of Multilingual Thesauri," in *The Semantic Web. Latest Advances and New Domains (Lecture Notes in Computer Science)*, vol. 9088, Springer International Publishing, 2015, pp. 38-53.

[3] World Wide Web Consortium (W3C), "SKOS Simple Knowledge Organization System Reference," World Wide Web Consortium, 18 August 2009. [Online]. Available: http://www.w3.org/TR/skos-reference/. [Accessed 22 March 2011].

[4] World Wide Web Consortium (W3C), "SKOS Simple Knowledge Organization System eXtension for Labels (SKOS-XL)," World Wide Web Consortium, 18 August 2009. [Online]. Available: http://www.w3.org/TR/skos-reference/skos-xl.html. [Accessed 22 March 2011].

[5] G. Hodge, Systems of Knowledge Organization for Digital Libraries: Beyond Traditional Authority Files, Washington, DC: Council on Library and Information Resources, 2000, p. 43.

[6] M. T. Pazienza, N. Scarpato, A. Stellato and A. Turbati, "Semantic Turkey: A Browser-Integrated Environment for Knowledge Acquisition and Management," *Semantic Web Journal*, vol. 3, no. 3, pp. 279-292, 2012.

[7] D. Griesi, M. T. Pazienza and A. Stellato, "Semantic Turkey - a Semantic Bookmarking tool (System Description)," in *4th European Semantic Web Conference (ESWC 2007)*, Innsbruck, Austria, 2007.

[8] A. Stellato, A. Turbati, M. Fiorelli, T. Lorenzetti, E. Costetchi, C. Laaboudi, W. Van Gemert and J. Keizer, "Towards VocBench 3: Pushing Collaborative Development of Thesauri and Ontologies Further Beyond," in *17th European Networked Knowledge Organization Systems (NKOS) Workshop. Thessaloniki, Greece, September 21st, 2017*, 2017.

[9] J. McCrae, D. Spohr and P. Cimiano, "Linking Lexical Resources and Ontologies on the Semantic Web with Lemon," in *The Semantic Web: Research and Applications (Lecture Notes in Computer Science)*, vol. 6643, Springer Berlin Heidelberg, 2011, pp. 245-259.

[10] J. P. McCrae, J. Bosque-Gil, J. Gracia, P. Buitelaar and P. Cimiano, "The OntoLex-Lemon Model: Development and Applications," in *Electronic lexicography in the 21st century. Proceedings of eLex 2017 conference.*, 2017.

[11] M. Fiorelli, A. Stellat, T. Lorenzetti, A. Turbati, P. Schmitz, E. Francesconi, N. Hajlaoui and B. Batouche, "Towards OntoLex-Lemon editing in VocBench 3," *AIDAinformazioni*, no. 3-4, 2018.

[12] P. Jain, P. Hitzler, P. Z. Yeh, K. Verma and A. P. Sheth, "Linked Data Is Merely More Data," in *Linked Data Meets Artificial*, AAAI Press, 2010, p. 82–86.

[13] B. McBride, «Jena: Implementing the RDF Model and Syntax Specification,» in *Semantic Web Workshop, WWW2001*, 2001.

[14] J. Broekstra, A. Kampman and F. van Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," in *The Semantic Web - ISWC 2002: First International Semantic Web Conference*, Sardinia, Italy, 2002.

[15] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak and S. Hellmann, "DBpedia - A crystallization point for the Web of Data," *Web Semantics: Science, Services*

and Agents on the World Wide Web, vol. 7, no. 3, p. 154–165, September 2009.

[16] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev and R. Velkov, "OWLIM: A family of scalable semantic repositories," *Semantic Web*, vol. 2, no. 1, pp. 33-42, 2011.

[17] M. Fiorelli, M. T. Pazienza, A. Stellato and A. Turbati, "CODA: Computer-aided ontology development architecture," *IBM Journal of Research and Development*, vol. 58, no. 2/3, pp. 14:1 - 14:12, March-May 2014.

[18] D. Ferrucci, "Unstructured Information Management Architecture (UIMA) Version 1.0," OASIS Standard, 2009.

[19] H. Cunningham, D. Maynard, K. Bontcheva and V. Tablan, "GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications," in *In Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia, 2002.

[20] M. T. Pazienza, A. Stellato and A. Turbati, "PEARL: ProjEction of Annotations Rule Language, a Language for Projecting (UIMA) Annotations over RDF Knowledge Bases," in *LREC*, Istanbul, 2012.

[21] M. Fiorelli, T. Lorenzetti, M. T. Pazienza, A. Stellato and A. Turbati, "Sheet2RDF: a Flexible and Dynamic Spreadsheet Import&Lifting Framework for RDF," in *Current Approaches in Applied Artificial Intelligence*, vol. 9101, M. Ali, Y. S. Kwon, C. Lee, J. Kim and Y. Kim, Eds., Springer International Publishing, 2015, pp. 131-140.

[22] F. Cotton, R. Cyganiak, R. Grim, D. W. Gillman, Y. Jaques and W. Thomas, "XKOS: An SKOS Extension for Statistical Classifications," in *Proceedings 59th ISI World Statistics Congress, 25-30 August 2013, Hong Kong (Session CPS203)*, 2013.

[23] C. Fellbaum, WordNet: An Electronic Lexical Database, Cambridge, MA: WordNet Pointers, MIT Press, 1998.

[24] M. Fiorelli, T. Lorenzetti, M. T. Pazienza and A. Stellato, "Assessing VocBench Custom Forms in Supporting Editing of Lemon Datasets," in *Language, Data, and Knowledge (Lecture Notes in Artificial Intelligence)*, vol. 10318, Springer, Cham, 2017, pp. 237-252.

[25] J. P. McCrae and C. Unger, "Design Patterns for Engineering the Ontology-Lexicon Interface," in *Towards the Multilingual Semantic Web*, P. Buitelaar and P. Cimiano, Eds., Springer Berlin Heidelberg, 2014, pp. 15-30.

[26] M. Heller, "REST and CRUD: the Impedance Mismatch," 29 January 2007. [Online]. Available: https://www.infoworld.com/article/2640739/application-development/rest-and-crud--the-impedance-mismatch.html. [Accessed 1 Febraury 2019].

[27] I. Bratko, Prolog Programming for Artificial Intelligence, Addison Wesley, 2001.

[28] E. Denti, A. Omicini and A. Ricci, "tuProlog: A Light-Weight Prolog for Internet Applications and Infrastructures," in *Practical Aspects of Declarative Languages (Lecture Notes in Computer Science)*, vol. 1990, Springer, Berlin, Heidelberg, 2001, pp. 184-198.

[29] M. Fiorelli, M. T. Pazienza, A. Stellato and A. Turbati, "Version Control and Change Validation for RDF Datasets," in *Metadata and Semantic Research (Communications in Computer and Information Science)*, vol. 755, E. Garoufallou, S. Virkus, R. Siatri and D. Koutsomiha, Eds., Springer, Cham, 2017, pp. 3-14.

[30] M. Fiorelli, M. T. Pazienza and A. Stellato, "Change management and validation for collaborative editing of RDF datasets," *International Journal of Metadata, Semantics and Ontologies*, vol. 12, no. 2-3, 2017.

[31] D. Hernández, A. Hogan and M. Krötzsch, "Reifying RDF: What Works Well With Wikidata?," in *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems, Bethlehem, PA, USA, October 11, 2015*, 2015.

[32] V. Nguyen, O. Bodenreider and A. Sheth, "Don't Like RDF Reification?: Making Statements About Statements Using Singleton Property," in *Proceedings of the 23rd International Conference on World Wide Web, April 7-11, 2014, Seoul, South Korea*, 2014.

[33] J. J. Carroll, C. Bizer, P. Hayes and P. Stickler, "Named Graphs, Provenance and Trust," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*, New York, NY, USA, 2005.

[34] R. Laurens and H. Rinkea, "The YASGUI family of SPARQL clients," *Semantic Web*, vol. 8, no. 3, pp. 373-383, 2017.

[35] J. David, J. Euzenat, F. Scharffe and C. Trojahn dos Santos, "The Alignment API 4.0," *Semantic Web Journal*, vol. 2, no. 1, pp. 3-10, 2011.

[36] World Wide Web Consortium (W3C), "Data Catalog Vocabulary (DCAT)," World Wide Web Consortium, 16 January 2014. [Online]. Available: http://www.w3.org/TR/vocab-dcat/.

[37] M. Dekkers, "Asset Description Metadata Schema (ADMS)," World Wide Web Consortium (W3C), 1 August 2013. [Online]. Available: http://www.w3.org/TR/vocab-adms/.

[38] K. Alexander, R. Cyganiak, M. Hausenblas and J. Zhao, "Describing Linked Datasets with the VoID Vocabulary (W3C Interest Group Note)," World Wide Web Consortium, 3 March 2011. [Online]. Available: http://www.w3.org/TR/void/. [Accessed 16 May 2012].

[39] M. Fiorelli, A. Stellato, J. P. Mccrae, P. Cimiano and M. T. Pazienza, "LIME: the Metadata Module for OntoLex," in *The Semantic Web. Latest Advances and New Domains (Lecture Notes in Computer Science)*, vol. 9088, Springer International Publishing, 2015, pp. 321-336.

[40] M. Fiorelli, M. T. Pazienza and A. Stellato, "An API for OntoLex LIME datasets," in *OntoLex-2017 1st Workshop on the OntoLex Model (co-located with LDK-2017)*, Galway, 2017.

[41] M. T. Pazienza, S. Sguera and A. Stellato, "Let's talk about our "being": A linguistic-based ontology framework for coordinating agents," *Applied Ontology, special issue on Formal Ontologies for Communicating Agents*, vol. 2, no. 3-4, pp. 305-332, 26 December 2007.

[42] M. T. Pazienza and A. Stellato, "An Environment for Semi-automatic Annotation of Ontological Knowledge with Linguistic Content," in *The Semantic Web: Research and Applications (Lecture Notes in Computer Science)*, vol. 4011, Springer, 2006, pp. 442-456.

[43] M. Fiorelli, M. T. Pazienza and A. Stellato, "A Meta-data Driven Platform for Semi-automatic Configuration of Ontology Mediators," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, 2014.

[44] E. Francesconi, M. Küster, P. Gratz and S. Thelen, "The Ontology-based Approach of the Publications Office of the

EU for Document Accessibility and Open Data Services," in *International Conference on Electronic Government and the Information Systems Perspective (EGOVIS 2015)*, Valencia, Spain, 2015.

[45] M. A. Musen, "The Protégé Project: A Look Back and a Look Forward," *AI Matters*, vol. 1, no. 4, pp. 4-12, 2015.

[46] T. Tudorache, C. Nyulas, N. F. Noy and M. A. Musen, "WebProtégé: A Collaborative Ontology Editor and Knowledge Acquisition Tool for the Web," *Semantic Web*, vol. 4, no. 1, pp. 89-99, 2013.

[47] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies," *Semantic Web*, vol. 2, no. 1, pp. 11-21, 2011.

[48] B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang, "HermiT: An OWL 2 Reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245-269, 2014.

[49] A. Hogan, J. Z. Pan, A. Polleres and Y. Ren, "Scalable OWL 2 Reasoning for Linked Data," in *Reasoning Web. Semantic Technologies for the Web of Data. Reasoning Web 2011 (Lecture Notes in Computer Science)*, vol. 6848, Springer, Berlin, Heidelberg, 2011, pp. 250-325.

[50] L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou and E. Simperl, Eds., "A Flexible API and Editor for SKOS," in *The Semantic Web: Research and Applications (Lecture Notes in Computer Science)*, vol. 5554, Springer, Berlin, Heidelberg, 2009, pp. 506-520.

[51] L. Halilaj, I. Grangel-González, G. Coskun, S. Lohmann and S. Auer, "Git4Voc: Collaborative Vocabulary Development Based on Git," *International Journal of Semantic Computing*, vol. 10, no. 2, pp. 167-191, June 2016.

[52] [Online]. Available: http://vocol.iais.fraunhofer.de/. [Accessed 11 2017].

[53] T. Tudorache, N. F. Noy, S. Tu and M. A. Musen, "Supporting Collaborative Ontology Development in Protégé," in *The Semantic Web - ISWC 2008 (Lecture Notes in Computer Science)*, vol. 5318, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin and K. Thirunarayan, Eds., Springer, Berlin, Heidelberg, 2008, pp. 17-32.

[54] M. Salvadores, P. R. Alexander, M. A. Musen and N. F. Noy, "BioPortal as a dataset of linked biomedical ontologies and terminologies in RDF," *Semantic Web*, vol. 4, no. 3, pp. 277-284, 2013.

[55] G. Mochón, E. M. Méndez and G. Bueno de la Fuente, "27 pawns ready for action: A multi-indicator methodology and evaluation of thesaurus management tools from a LOD perspective," *Library Hi Tech*, vol. 35, no. 1, pp. 99-119, 2017.

[56] C. Mader, B. Haslhofer and A. Isaac, "Finding Quality Issues in SKOS Vocabularies," in *Theory and Practice of Digital Libraries (Lecture Notes in Computer Science)*, vol. 7489, Springer, Berlin, Heidelberg, 2012, pp. 222-233.

[57] M. Fiorelli, M. T. Pazienza, S. Petruzza, A. Stellato e A. Turbati, «Computer-aided Ontology Development: an integrated environment,» in *New Challenges for NLP Frameworks 2010 (held jointly with LREC2010)*, La Valletta, Malta, 2010.

[58] M. Fiorelli, R. Gambella, M. T. Pazienza, A. Stellato and A. Turbati, "Semi-automatic Knowledge Acquisition through CODA," in *Modern Advances in Applied Intelligence - 27th International Conference on Industrial Engineering and Other Applications of Applied Intelligent System, IEA/AIE 2014*, Kaohsiung, Taiwan, 2014.

[59] World Wide Web Consortium (W3C), "Shapes Constraint Language (SHACL)," World Wide Web Consortium (W3C), 20 July 2017. [Online]. Available: https://www.w3.org/TR/shacl/. [Accessed 13 November 2017].

[60] E. Montiel-Ponsoda, J. McCrae and P. Cimiano, "Collaborative semantic editing of linked data lexica," in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12). Istanbul Lüfti Kirdar Convention & Exhibition Centre, Turkey, 21-27 May 2012*, 2012.

[61] M. Rico and C. Unger, "Lemonade: A Web Assistant for Creating and Debugging Ontology Lexica," in *Natural Language Processing and Information Systems (Lecture Notes in Computer Science)*, vol. 9103, Springer, Cham, 2015, pp. 448-452.

[62] A. Bellandi, E. Giovannetti, S. Piccini and A. Weingart, "Developing LexO: A Collaborative Editor of Multilingual Lexica and Termino-ontological Resources in the Humanities," in *Proceedings of Language, Ontology, Terminology and Knowledge Structures Workshop (LOTKS 2017),co-located with the 12th International Conference on Computational Semantics (IWCS), 19 September 2017 Montpellier*, 2017.

[63] A. Bellandi, E. Giovannetti and A. Weingart, "Multilingual and Multiword Phenomena in a lemon Old Occitan Medico-Botanical Lexicon," *Information*, vol. 9, no. 3, 2018.

[64] A. Gonzales-Aguilar, M. Ramírez-Posada and D. Ferreyra, "TemaTres: software para gestionar tesauros," *El profesional de la información*, vol. 21, no. 3, pp. 319-325, 2012.

[65] F. Bond and K. Paik, "A survey of wordnets and their licenses," in *Proceedings of the 6th Global WordNet Conference (GWC 2012). Matsue, Japan, January, 9-13, 2012* .