OntoWiki – An Authoring, Publication and Visualization Interface for the Data Web

Philipp Frischmuth [*], Michael Martin, Sebastian Tramp, Thomas Riechert, Sören Auer,
University of Leipzig, Institute of Computer Science, AKSW Group, Augustusplatz 10, D-04009 Leipzig, Germany
E-mail: {lastname}@informatik.uni-leipzig.de

Abstract. OntoWiki is a front-end application for the Semantic Data Web, which was originally developed to support distributed knowledge engineering scenarios. Due to its extensibility it also serves as a development framework for knowledge intensive applications. On the surface, OntoWiki is a generic user interface for arbitrary RDF knowledge graphs. It supports the navigation through RDF knowledge bases using SPARQL-generated lists, tables and trees (e.g. class trees and taxonomies). All resources are automatically represented as hyperlinks and backlinks are created whenever feasible, thus enabling users to easily traverse entire knowledge graphs. Since all collections of resources displayed in OntoWiki are generated by SPARQL queries, they can be further refined by applying additional filters. In order to explore large datasets, a comprehensive statistical data management and visualization method was integrated. We render an architectural overview and explain the navigation, exploration and visualization features provided for RDF based information and derived statistics. To showcase the versatility of OntoWiki and its various deployments in the Linked Data Web, we describe some large-scale use cases in the domains of enterprise data integration, governmental data publication and digital humanities. This article is the first comprehensive presentation of OntoWiki summarizing the advancements after the initial publication in 2006 [4].

Keywords: Linked Data, Data Web, RDF, SPARQL, Wiki

1. Introduction

Despite its recent increase in popularity, the Semantic Data Web is still difficult to interact with end users. There is a large amount of structured data adhering to the RDF data model being published on the Web. However, it is cumbersome to discover, access and explore this data. Also, the Data Web is currently still rather a Read Web than a Read-Write Web – the vast amount of information is published by a relatively small amount of publishers [3]. One approach for making the Semantic Web a Read-Write Web are Semantic Wikis.

From Semantic Wikis to Semantic Data Wikis. Semantic Wikis are an extension to conventional, text-based Wikis. While in conventional Wikis pages are stored as blocks of text using a special Wiki markup for structuring the display of the text and adding links to other pages, semantic Wikis aim at adding rich structure to the information itself. To this end, two initially orthogonal approaches have been used: a) extending the markup language to allow semantic annotations and links with meaning or b) building the Wiki software directly with structured information in mind. Nowadays, both approaches have somewhat converged, for instance *Semantic MediaWiki* [24] also provides forms for entering structured data (see Figure 1). Characteristics of both approaches are summarized in Table 1 for the two prototypical representa-

---

[*]Corresponding author. E-mail: frischmuth@informatik.uni-leipzig.de.

tives of both approaches, i.e. Semantic MediaWiki and OntoWiki.

Table 1

Conceptual differences between Semantic MediaWiki and OntoWiki.

	Semantic MediaWiki	**OntoWiki**
Managed entities	Articles	Resources
Editing	Wiki markup	Forms
Atomic element	Text blob	Statement

Extending Wikis with Semantic Markup. The benefit of a Wiki system comes from the amount of interlinking between Wiki pages. Those links clearly state a relationship between the linked-to and the linking page. However, in conventional Wiki systems this relationship cannot be made explicit. Semantic Wiki systems therefore add a means to specify typed relations by extending the Wiki markup with semantic (i.e. typed) links. Once in place, those links form a knowledge base underlying the Wiki which can be used to improve search, browsing or automatically generated lists and category pages. Examples of approaches for extending Wikis with semantic markup can be found in [24,36,6,33,37]. They represent a straightforward combination of existing Wiki systems and the Semantic Web knowledge representation paradigms. Yet, we see the following obstacles:

Usability: The main advantage of Wiki systems is their unbeatable usability. Adding more and more syntactic possibilities counteracts ease of use for editors.

Redundancy: To allow the answering of real-time queries to the knowledge base, statements have to be additionally kept in a triple store. This introduces a redundancy, which complicates the implementation.

Evolution: As a result of storing information in both Wiki texts and triple store, supporting evolution of knowledge is difficult.

Wikis for Editing Structured Data. In contrast to text-based systems, Wikis for structured data – also called Data Wikis – are built on a structured model of the data being edited. The Wiki software can be used to add instances according to the schema or (in some systems) edit the schema itself. OntoWiki, as a representative of this class, bases its data model on RDF. This

way, both schema and instance data are represented using the same low-level model (i.e. statements) and can therefore be handled identically by the Wiki.

OntoWiki - a Semantic Data Wiki. OntoWiki started as an RDF-based data wiki with emphasis on collaboration but has meanwhile evolved into a comprehensive framework for developing Semantic Web applications [21]. This involved not only the development of a sophisticated extension interface allowing for a wide range of customizations but also the addition of several access and consumption interfaces allowing OntoWiki installations to play both a provider and a consumer role in the emerging Web of Data. OntoWiki is inspired by classical Wiki systems, its design, however, (as mentioned above) is independent and complementary to conventional Wiki technologies. In contrast to other semantic Wiki approaches, in OntoWiki text editing and knowledge engineering (i.e. working with structured knowledge bases) are not mixed. Instead, OntoWiki directly applies the Wiki paradigm of "making it easy to correct mistakes, rather than making it hard to make them" [25] to collaborative management of structured knowledge. This paradigm is achieved by interpreting knowledge bases as *information maps* where every node is represented visually and interlinked to related resources. Furthermore, it is possible to enhance the knowledge schema gradually as well as the related instance data agreeing on it. As a result, the following features characterize OntoWiki:

Intuitive display and editing of instance data is provided in generic ways, yet enabling domain-specific presentation of knowledge.

Semantic views allow the generation of different views and aggregations of the knowledge base.

Versioning and evolution provides the opportunity to track, review and roll-back changes selectively.

Semantic search facilitates easy-to-use full-text searches on all literal data, search results can be filtered and sorted (using semantic relations).

Community support enables discussions about small information chunks. Users are encouraged to vote about distinct facts or prospective changes.

Online statistics interactively measures the popularity of content and activity of users.

Semantic syndication supports the distribution of information and their integration into desktop applications.

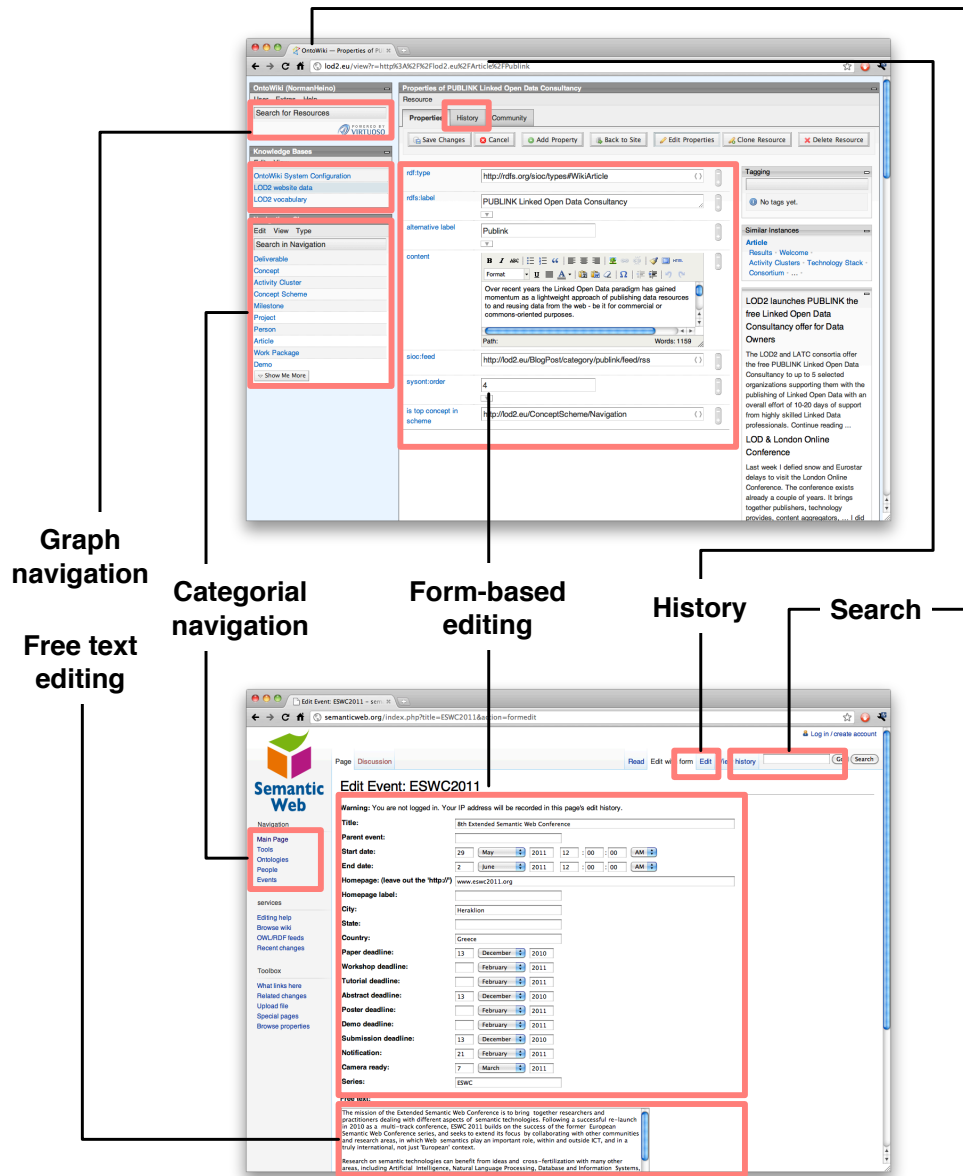OntoWiki enables the easy creation of highly structured content by distributed communities. The fol-

Fig. 1. Comparison of Semantic MediaWiki and OntoWiki GUI building blocks.

lowing points summarize some limitations and weaknesses of OntoWiki and thus characterize the application domain:

Environment: OntoWiki is a Web application and presumes all collaborators work in a Web environment, possibly distributed.

Usage Scenario: OntoWiki focuses on knowledge engineering projects where a single, precise usage scenario is either initially (yet) unknown or not (easily) definable.

Reasoning: Reasoning services are not in the primary focus of application.

Application scenarios. One of the main characteristics is the versatility of OntoWiki. In its 7 year life-time, there have been numerous deployments and a number of large-scale comprehensive use cases in many different domains. Three complementary use cases involving stakeholders from very different domains are presented in this article:

Enterprise Data Integration. Data integration in large enterprises and organizations is a crucial but at the same time costly, long lasting and challenging problem. While business-critical information is often already gathered in integrated information systems, such as ERP, CRM and SCM systems, the integration of these systems itself as well as the integration with the abundance of other information sources is still a major challenge. We present how the use of Semantic Wiki technology facilitates the integration of enterprise data. In particular, existing enterprise taxonomies which capture the key concepts and terms of an organization can evolve using OntoWiki into comprehensive enterprise knowledge hubs interlinking a large number of knowledge and data sources in the enterprise.

Open Data Publication and Visualization. Recently, we have seen a rapid growth of open data catalogs being made available on the Web. The data catalog registry datacatalogs.org, for example, already lists 285 data catalogs worldwide. Many of the datasets published in these data catalogs, however, are cumbersome to explore and visualize. With the generic OntoWiki arbitrary datasets can be browsed. For statistical data, which is one of the most important open data types, we developed with CubeViz a specific extension to OntoWiki facilitating the user-directed visualization of data in various chart and diagram types. We report about the deployment of OntoWiki at the European Commissions's Open Data portal.

Digital Humanities. With the Catalogus Professorum Lipsiensium (CPL) we developed an adaptive, semantics-based knowledge engineering application based on OntoWiki for prosopographical knowledge. In prosopographical research, historians analyze common characteristics of historical groups by studying statistically relevant quantities of individual biographies. Researchers from the Historical Seminar at University of Leipzig created a prosopographical knowledge base about the life and work of professors in the 600 years history of University of Leipzig ranging from the year 1409 till 2009 – the *Catalogus Professorum Lipsiensium*. In order to enable historians to collect, structure and publish this prosopographical knowledge the *Catalogus Professorum Model* was developed and incrementally refined using OntoWiki over a period of three years. As result, a group of 10 historians supported by a much larger group of volunteers and external contributors collected information about 2,000 professors, 15,000 associated periods of life, 400 institutions and many more related entities.

Structure of the article. This article is structured as follows: We give an overview on OntoWiki's architecture in Section 2. The discovery and exploration features including key user interface elements and the visual representation of semantic content is presented in Section 3. In Section 4 we detail the authoring and content management functionality. We describe techniques implemented in OntoWiki facilitating machine-consumption and data integration in Section 5. Three complementary use cases are presented in Section 6. Finally, we conclude with an outlook on future work in Section 7.

2. Architectural Overview

This section provides an architectural overview on OntoWiki. We start with an explanation of the general architecture as depicted in Figure 2 and describe the key building blocks (denoted with bold frames in the figure). We organized the architecture in the figure in three layers comprising a *backend*, *application* and *frontend layer* as well as a vertical meta layer. The meta layer encapsulates the ecosystem of third party extensions available for use within the remaining three layers.
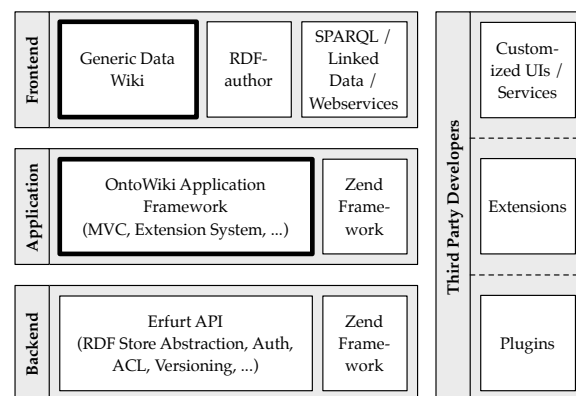

Fig. 2. The OntoWiki architecture divided into three layers: backend layer, application layer and frontend layer. An additional vertical meta layer depicts the extension ecosystem of OntoWiki.

The backend layer comprises the *Erfurt API*[1] and tangentially (through usage and extension of functionality) the *Zend Framework*[2], a stable and well established web application framework for PHP.

---

[1] http://aksw.org/Projects/Erfurt
[2] http://framework.zend.com/

The Erfurt API supports the development of semantic web applications more generally. It is built and maintained parallel to OntoWiki and provides functionality for persisting RDF quads (RDF triples and context information) as well as retrieving those via the SPARQL query language [18]. The storage abstraction layer of the Erfurt API facilitates the usage of different RDF stores, for example, *Virtuoso* [15] or the built-in RDF store that persists triples in a relational database and translates SPARQL queries into corresponding SQL queries. In addition, Erfurt API comprises authentication and access control features, which maintain user related access control information in a separate RDF system knowledge base. Further features provided by Erfurt API include:

- support for versioning of RDF data,
- a SPARQL query cache [28] that significantly accelerates complex queries,
- a plug-in environment based on events as well as
- a lightweight resource wrapper mechanism for accessing and converting arbitrary resources (e.g. from specific CSV files or REST APIs identified via URIs) into RDF resources.

On top of this layer the application layer involves the *OntoWiki Application Framework* [21], which on the one hand again uses and extends Zend Framework functionality (e.g. for the Model-View-Controller infrastructure) and on the other hand uses Erfurt API (e.g. for accessing a triple store).

The frontend layer consists of the generic Data Wiki, which is ontology and vocabulary agnostic and hence can be used out-of-the-box with arbitrary RDF knowledge bases. Furthermore, it includes *RDFauthor* [39] to provide user-friendly editing forms, as well as other access interfaces like a SPARQL protocol [16] conform endpoint, and a Linked Data [9] endpoint.

Third party developers are enabled and encouraged to extend the functionality of OntoWiki on all three layers. On the backend level plug-ins can be developed that listen to certain events (e.g. changes on the triple store). Multiple extension mechanisms are available on the application level to add new functionality or change existing behavior, which result in customized user interfaces and services on the frontend level.

2.1. Generic Data Wiki

OntoWiki can be used out-of-the-box as a generic tool for publication, exploration, authoring and maintenance of arbitrary RDF knowledge bases. For this purpose it provides generic methods and views, which do not require any tailoring to the domain concerned. We refer to it as a Wiki, because it adheres to the Wiki principles [25]:

1. Although access control is supported, by default everyone can contribute changes and participate in the development/evolution of a knowledge base.
2. Since RDF allows schema information and instance data to be mixed, content and structure of a knowledge base can be edited in tight relation.
3. All activities are tracked, such that it becomes easy to correct mistakes.
4. Changes can be discussed online on resource level.

OntoWiki is based solely on the RDF data model and consequently focuses on structured information instead of textual content with some added semantics.

The resource view and the list view are the two generic views that are included in the OntoWiki core. The resource view is generally used for displaying a description with all known information about a resource. The list view represents a set of resources, for example, instances of a certain concept. These two views are sufficient for browsing and editing all information contained in a knowledge base in a generic way.

Figure 3 depicts the typical workflow when working with OntoWiki as a generic tool. When a user visits the homepage of an OntoWiki deployment she is presented with a list of existing knowledge bases (cf. 1 in Figure 3). After selecting a knowledge base she is then presented with a hierarchy (obtained from `rdfs:subClassOf` statements) of classes, which are themselves instances of `owl:Class` or `rdfs:Class`. By selecting one of these classes, the user receives a list of resources that are instances of this class. In Figure 3 the class `Person` has been selected and yields a list of persons being either a direct instance of `Person` or one of its subclasses. In order to achieve this, OntoWiki applies basic RDFS [11] reasoning for computing the transitive closure of the `rdfs:subClassOf` relation automatically. Once a list of resources is retrieved, it can be extended by selecting additional properties that should be displayed (cf. 3 in Figure 3) and refined by selecting facet restrictions (cf. 4 in Figure 3). The user interface elements for expanding and restricting the current view adapt both automatically according to the resources currently
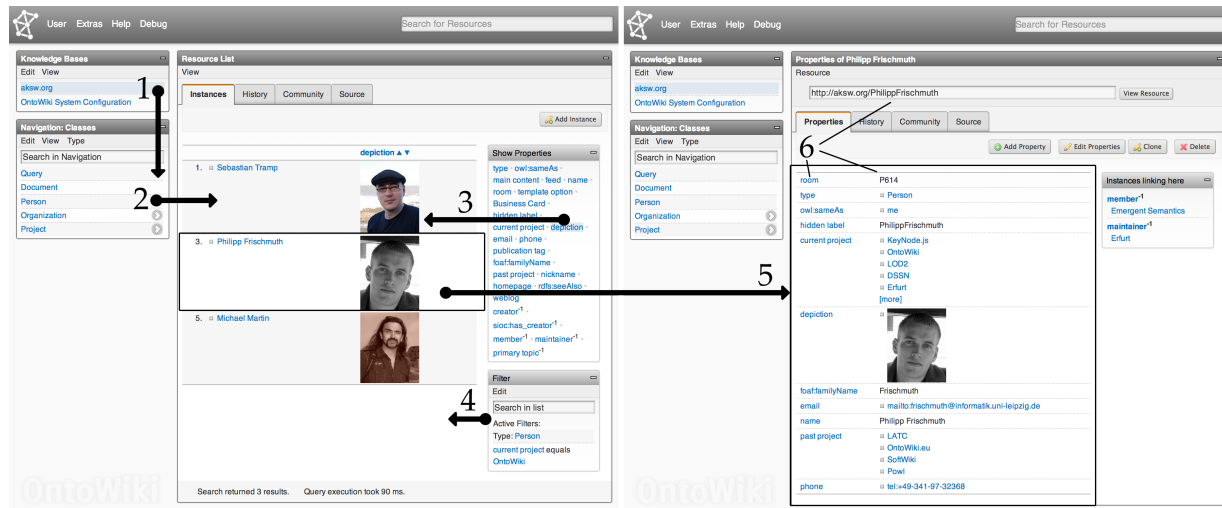
Fig. 3. Screenshot of OntoWiki with typical workflow: 1) selection of a knowledge base; 2) selection of a class; 3) selection of additional properties to be shown as columns in the list; 4) further restriction of the resources in the list; 5) selection of a resource redirects the user to a generic resource details view; 6) representation of RDF triples in the user interface as resource attribute value notation.

contained in the list (i.e. the properties offered to expand the tabular view and the facets are dynamically recomputed). In Figure 3 the list of instances of the class `Person` was further filtered, such that it only contains those resources, that currently work on the OntoWiki project. After selecting an instance from the list the user is directed to the generic resource view, which is depicted in the right part of Figure 3. Here it is possible to view all information available for the selected resource as well as to manage (i. e. insert, edit and update) it.

2.2. OntoWiki Application Framework

OntoWiki was initially developed as a generic tool for social, semantic collaboration on RDF knowledge bases [4]. Although we still refer to it as a generic Data Wiki (as described in Section 2.1), most APIs used to render its functionality are also available to third-party developers. The provided points of contact allow developers to extend, customize and tailor OntoWiki in several ways. In fact larger parts of the functionality included in a default OntoWiki setup is realized using those extension mechanisms only. Thus, we also refer to OntoWiki as the *OntoWiki Application Framework (OAF)* [21].

The building blocks of the OAF are the Model-View-Controller (MVC), the Linked Data infrastructure and the extension system. The former ensures that all incoming requests are dispatched to an appropriate controller providing a certain functionality (possi-

bly hosted by an extension). A controller groups handlers for different kinds of requests into actions, which then use different types of model classes and model helper classes to prepare content to be finally rendered using view templates. One of OntoWiki's most outstanding features is that it automatically displays human-readable representations of resources instead of URI strings. The naming properties it uses are configurable and SPARQL queries that test all those properties can be quite complex. OntoWiki therefore provides a model helper class that handles the nitty-gritty details of fetching titles of resources.

The Linked Data infrastructure provided by OntoWiki ensures that requested resources (identified by URIs) that do not relate to a controller/action pair and that have a description in the RDF store get served in the most appropriate format. Thus, a user providing a URI within the address bar of his browser would get the HTML representation while another Linked Data capable tool would get a representation in an RDF serialization format (e.g. RDF/XML [7]).

The OntoWiki extension system consists of four main extension types:

Plug-ins are the most basic, yet most flexible types of extensions. They consist of arbitrary code that is executed on certain events. OntoWiki uses plug-ins to render certain information resources (e.g. images, videos or documents on the web) with the appropriate tags in the HTML output (cf. 1 Figure 4).
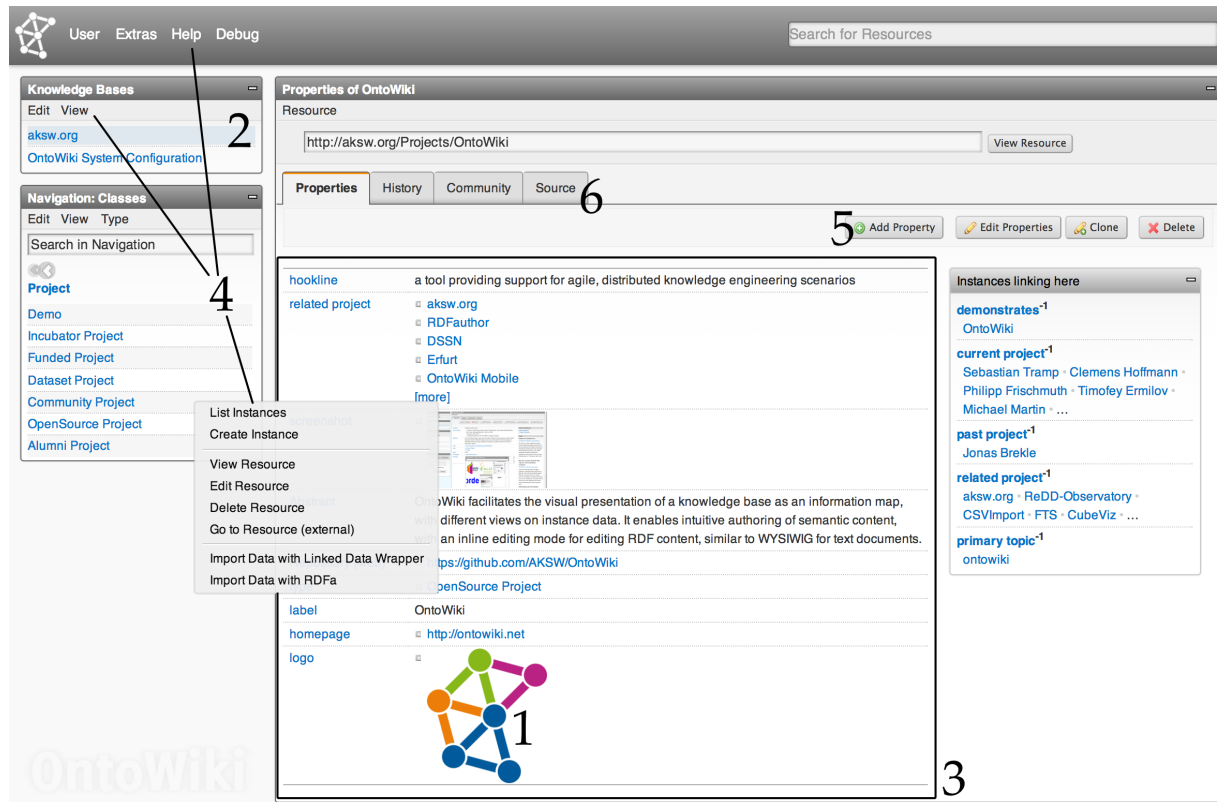
Fig. 4. Screenshot of OntoWiki with: 1) image rendered by plugin, 2) module window, 3) main content provided by component, 4) different types of menus, 5) toolbar and 6) navigation.

Wrappers are lightweight plug-ins that enable developers to provide data in the RDF format for arbitrary resources. A social web application based on OAF for example would use a wrapper to fetch data from Twitter via its API and translate the JSON data into RDF with the help of the *SIOC vocabulary* [10].

Modules display little windows that provide additional user interface elements with which the user can affect the main window's content. By default OntoWiki displays a class hierarchy next to the main content to enable users to quickly navigate through a knowledge base. For this purpose a module is used to render the content (cf. 2 in Figure 4).

Components are pluggable MVC controllers to which requests can be dispatched. Usually but not necessarily, components provide the main window's content (cf. 3 in Figure 4) and, in that case, can register with the navigation to be accessible by the user. In other cases components can function as controllers that provide services without a con-

crete UI representation (e.g. serve asynchronous requests). A typical example for such an extension is the map extension[3]. It includes a controller that renders resources on a map using a configurable set of properties for the geolocation.

In addition to these facilities a few connecting factors exist in the user interface, which allow extensions to hook into:

Menus and context menus exist throughout the user interface of OntoWiki (cf. 4 in Figure 4). Extensions are enabled to add as well as replace entries in those menus.

Toolbar is a centrally managed UI element that ensures a consistent user interface in all views. It is displayed above the main content as depicted under 5 in Figure 4. Extensions can append or prepend buttons and separators to the toolbar or disable it if not applicable.

---

[3] https://github.com/AKSW/map.ontowiki

Navigation Bar in OntoWiki by default is a tab bar displayed above the main content (and the toolbar if enabled). A typical navigation is shown under 6 in Figure 4. Components can register one or more actions with the navigation, which will result in additional tabs or disable the navigation.

Messages represent user notifications and can have a message text as well as a type (success, info, warning, error). OntoWiki keeps a stack of these messages and extensions can add messages to this stack, which will then be shown in the upper part of the main content.

Furthermore it is possible to customize the user interface with the help of themes and adapt it to the target audience using localizations. Localizations of OntoWiki are already available for a set of common languages (including English, German, Russian and Chinese). Extensions can also provide their own localizations for output that they generate.

3. Discovery and Exploration

Table 2

Categorization of applicable UI element types by level of data awareness – OntoWiki with its extensibility provides support for all levels of data awareness.

Data Awareness	Applicable UI Element Types	Example UI Elements
0 (none)	generic	lists, resource views (Section 3.1)
1 (partial)	common	resource titles, images, weblinks, maps (subsubsection 3.2.1)
2 (structural)	common	hierarchies (classes, SKOS, properties), statistical charts (subsubsection 3.2.2 and subsubsection 3.2.3)
3 (full)	custom-built	custom pages (subsubsection 3.2.4) and forms

OntoWiki provides a wide range of possibilities to discover and explore RDF knowledge bases. Table 2 shows a categorization of applicable UI element types by level of data awareness. We introduce three main UI element types:

Generic UI elements can be applied to arbitrary RDF data.

Common user interface elements are reusable in a variety of domains. Depending on the characteristics of the element very little (e.g. utilized properties) up to more profound (e.g. employed meta-vocabularies) knowledge about the data is required.

Custom-built UI elements are tailored to a particular domain and thus only operate on a very narrow set of data.

Those element types are related to four awareness levels:

Level 0 denotes that *no* knowledge about the data needs to exist (other than that it is represented using RDF). Thus only generic UI elements are applicable in this situation. Since OntoWiki is a generic tool it provides two key user interface elements for generic data, namely resource lists based on SPARQL queries and table-based resource views.

Level 1 requires that *partial* knowledge about the data is available. For example, most datasets typically contain a human readable title for resources. If the utilized properties are known, the user interface can adapt accordingly and render a title instead of a URI.

Level 2 represents the existence of *structural* knowledge about the data. Often resources are grouped into classes using meta-vocabularies like RDFS or OWL [29] or concepts are related to each other employing SKOS [30]. Another example is statistical metadata that is added on top of instance data using a vocabulary like DataCube [14].

Level 3 denotes that *full* knowledge about the data is available. Hence fully customized user interface elements become feasible in order to support the users, such as highly adapted (and possibly styled) HTML renderings and simplified forms.

OntoWiki with its extensibility has support for all levels of data awareness by providing appropriate user interface elements either out of the box or with the help of third party extensions. In the remainder of this section we describe the different capabilities of the tool in this regard.

3.1. Key User Interface Elements (Human Consumption)

OntoWiki provides two key user interface elements to support the visualization of generic RDF data,

namely a list and a resource view. As those were already briefly introduced in the overview section (cf. Section 2), Figure 3 depicts both of them.

3.1.1. Generic SPARQL-based Lists

Lists are a central UI element within the OntoWiki tool. In most cases users start browsing a dataset either by

– issuing a keyword search,
– selecting a resource in the navigation interface or
– constructing a SPARQL query.

In the first two cases a SPARQL select query is created with appropriate filter conditions. When a user provides a SPARQL query directly, the query needs to satisfy some constraints. It must be a select query and the first column of the result set must contain URIs. In all cases a SPARQL *base query* exists that is persisted until the user creates/selects a new list. In the meantime it is likely that the base query is modified several times in order to further refine the list.
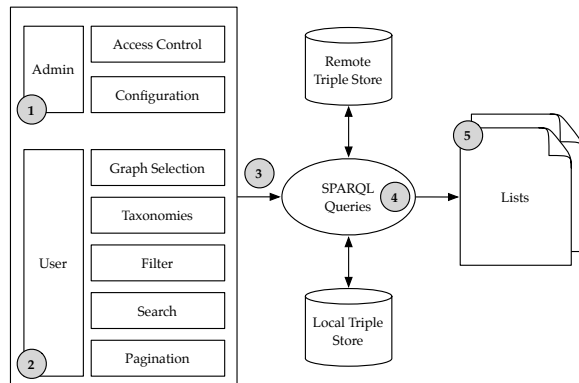
Fig. 5. The process of rendering a list from arbitrary RDF data within OntoWiki: Administrative facets (1) and facets controlled by the user (2) influence the creation of SPARQL queries (3), which are then executed against local or remote endpoints (4) and finally rendered as HTML tables (5).

Figure 5 depicts the facets that influence the SPARQL queries that finally result in lists rendered as HTML tables. In the first step an administrator can restrict access to certain knowledge bases, such that all SPARQL queries issued to the OntoWiki deployment in question will only operate on allowed graphs. This is achieved by filtering and explicitly assigning FROM clauses to all queries. An administrator can also influence certain base queries by configuration. The navigation interface for example is highly customizable not only in regard to what is shown (e.g. a class hierar-

chy), but also in regard to what happens when an item is selected.

Primarily however the user affects the base query for a list:

– By selecting a certain knowledge base, the FROM clause of the query is filtered again.
– Adding a constraint that further restricts a list (via the filter module) also adds a suitable FILTER condition to the base query.
– Issuing a free text search results in the addition of a FILTER(regex(...)) condition.
– By selecting a specific range of a list (pagination), which leads to a LIMIT, OFFSET combination set for the base query.

Afterwards a second SPARQL query (*value query*) is created, which is responsible for fetching values for additionally selected columns. Both queries are then, depending on the configuration, executed against a local or a remote triple store. The results are combined and finally rendered into a generic yet dynamic list.

Since the generic lists utilized within OntoWiki are solely SPARQL based and no assumptions regarding the underlying data and applied vocabularies are made, these lists perform well on arbitrary RDF datasets.

3.1.2. Generic (table-based) resource views

The second key user interface element for generic data in OntoWiki is the table-based resource view. It is capable of rendering arbitrary resources, since it exploits the simple structure of RDF statements. In order to prepare the rendering, all statements contained in the currently active knowledge base are selected, that match the resource to be shown in the subject position. The main table then consists of two columns:

– one for the property (predicate of the statement) and
– a second column for the value(s) (object of the statement).

This is achieved by issuing a simple SPARQL query. To maintain the clarity, values are grouped by properties and only a configurable number of values are included in the final rendering. If that number is exceeded, a *show me more* link is included, which causes the creation of a new generic list. By default all resources, which include resource values and properties are rendered as internal links, i.e. as links to the generic resource view.

The main table only contains values where the resource is the subject of a suitable statement. The

generic nature of RDF however implies that useful information for a resource is not necessarily encoded that way. In some situations the resource may be related to another resource the other way around, i.e. occurring in the object position of statements (e.g. containment relations are often modeled this way). For this purpose an additional interface is integrated in the resource view, which displays incoming links in a small window.

3.2. *Visual Representation of Semantic Content*

In this section we present the visualization capabilities of OntoWiki with regard to the remaining data awareness levels in ascending order.

3.2.1. *Visualization Helper and Plugins*

Naturally human users and especially non-technical users do not remember URIs well and user interfaces quickly become confusing, if they get polluted with them. OntoWiki therefore hides URIs from users whenever feasible, displaying a title instead that at its best fits the demands of the users.

Title Helper. Since titles for resources are required in various situations, OntoWiki includes a helper class that centrally handles this important functionality. This has the following advantages:

– resources are represented consistently throughout the application,
– other parts of the tool (especially extensions) do not have to bother with the details of retrieving titles and
– the results of the retrieval can be cached in order to improve performance.

By default the title helper checks for titles using properties from well-known vocabularies like SKOS, DCTerms[4], FOAF[5], DOAP[6], SIOC and RDFS. It also prefers those values that are tagged with the users preferred language. Since many datasets already include descriptions using at least one of the above vocabularies, the title helper often works out-of-the-box with arbitrary RDF data. If a dataset utilizes custom properties to represent titles, OntoWiki can be easily configured to use those properties instead.

---

[4]http://dublincore.org/documents/dcmi-terms/
[5]http://xmlns.com/foaf/spec/
[6]https://github.com/edumbill/doap/wiki

When working with RDF data it is not always guaranteed that properties are used consistently within a knowledge base. For example, a graph can consist of multiple subgraphs that originate from different sources. To solve this issue the title helper maintains an ordered list of possible title properties. Higher ranked properties will be preferred when a value is available and lower ranked properties are used otherwise. If no title can be determined at all, the title helper still tries to return something meaningful. For example, HTTP URIs are decomposed into a namespace part and a local name. If the namespace is well-known and a prefix was defined for it (e.g. `rdfs`[7]), the title helper will return the local name appended to the prefix followed by a colon.

Plugins. In addition to displaying a title for all resource URIs, OntoWiki by default renders the resources as internal links to the generic resource view. However, this is not always the desired behavior as illustrated with the help of the FOAF vocabulary:

– `foaf:homepage` is used to relate something to a homepage about it. Since OntoWiki is a web based application, users may expect to be presented with the target homepage instead of an OntoWiki rendering of the resource.
– `foaf:mbox` is used to relate an agent to a mailbox identified by a `mailto:` URI. Users probably want to open their personal email client when following such links.
– `foaf:depiction` relates something to an image about it. It is very likely that users will very much appreciate being presented with the actual image instead of a textual link.

OntoWiki contains plugins for all three of the above cases. These plugins are enabled by default and configured to work with many datasets without further modification. The `weblink` plugin renders resources as external links to the target resource. The same applies for the `mailtolink` plugin, but a `mailto:` protocol string is prepended iff not existent in the value. Finally the `imagelink` plugin employs the HTML `<img>` tag to make images visible to the users.

Other cases require more sophisticated interventions than replacing the rendering of a single value only. Geographic coordinates for example are often encoded with two statements, one to represent the longitude and another to encode the latitude. Here the OntoWiki map

---

[7]http://www.w3.org/2000/01/rdf-schema#

extension adds, amongst other things, a rendering of the geolocation on a map.

3.2.2. Navigation Component

RDF knowledge bases are typically structured by

- grouping instances into classes and arranging those classes with a subclass relationship,
- describing concepts and their relationship to other broader or more narrow concepts or
- utilizing a set of properties to describe entities of the domain of interest.

The navigation component is a powerful OntoWiki extension that is able to extract the structure of knowledge bases in the above mentioned situations and thus facilitate the navigability of datasets. As already described in Section 2.1, the navigation box (cf. left side of Figure 4) renders a class hierarchy by default. An arrow next to a class name indicates, that the class has at least one subclass. Following this link will result in an updated list showing only those subclasses. When a class is selected from the list, all instances of this class will be presented to the user in the main window.

Through an entry in the menu of the box the user can select other configurations. The *SKOS configuration* shows a list of `skos:Concept` instances and utilizes the `skos:broader` and `skos:narrower` relationships to produce a hierarchy. Depending on the configuration the result of selecting a concept is either

- a list of resources (e.g. resources, which are related to the concept via `dcterms:subject`) or
- a rendering of the concept itself (via the generic resource view).

Another example to showcase the flexibility of the navigation component is the *Properties configuration*. It shows a hierarchy of properties that are used within a knowledge base (via `rdfs:subPropertyOf`). When a user selects a property, a list with resources is created again. In this case only those resources are contained, that make use of the property. Additionally the selected property is added as a column to the list, such that the user can immediately work with the values. The navigation extension is highly configurable and can be tailored in various ways to better represent the data.

3.2.3. CubeViz - Discovery of statistical Linked Data

In the recently published open data portal of the European Commission[8], more than 5.700 statistical datasets are listed. Hence, currently almost 95% of the datasets are statistical data. Most of these datasets are published using CSV, XML or similar representations, but there are several efforts to convert and publish them as RDF as can be seen by the example of the RDF version of Eurostat data[9]. Certainly, it is possible to develop web applications specifically adopted to particular datasets (e.g. The Digital Agenda Scoreboard[10]) or to use diverse spreadsheet tools to browse over the data. However, this is time-consuming and cost-intensive, as well not cumbersome for data consumers. In the following we describe the *RDF DataCube vocabulary*, provide a state-of-the-art overview in the field of representing statistics in RDF, and introduce *CubeViz*[11], an OntoWiki extension component for exploring statistics represented using this vocabulary.

The DataCube Vocabulary. The representation of statistics in RDF started with *SCOVO* [19,13] and continued with its successor the RDF Data Cube Vocabulary [14], which has been established as a common standard. This vocabulary is compatible with the Statistical Data and Metadata eXchange (SDMX, [23]) and is being increasingly adopted. SDMX is an initiative started in 2001 to foster standards for the exchange of statistical information. The SDMX sponsoring institutions are the Bank for International Settlements, the European Central Bank, Eurostat, the International Monetary Fund (IMF), the Organisation for Economic Co-operation and Development (OECD), the United Nations Statistics Division and the World Bank. The SDMX message formats have two basic expressions, SDMX-ML (using XML syntax) and SDMX-EDI (using EDIFACT syntax and based on the GESMES/TS statistical message). Experiences and best practices regarding the publication of statistics on the Web in SDMX have been published by the United Nations [31] and the Organization for Economic Co-operation and Development [32].

In brief, to encode structural information about statistical observations, the RDF Data Cube vocabulary contains a set of concepts, such as `qb:DataStructureDefinition`[12] and `qb:DataSet`. Every statistical observation has to be encoded as `qb:Observation` and must be linked to a specific resource of type `qb:DataSet`. Existing resources

---

[8] `http://open-data.europa.eu/`

[9] `http://eurostat.linked-statistics.org/`

[10] `http://digital-agenda-data.eu/`

[11] `http://aksw.org/Projects/CubeViz`

[12] qb is a prefix used as an abbreviated form for `http://purl.org/linked-data/cube#`

typed as `qb:DataSet` usually have one relation to an instance of type `qb:DataStructureDefinition`, which is used to declare the structure of each observation that is linked to the specific dataset. Such an observation structure is given by a set of component properties used to encode dimensions, attributes and measures. Each property is an instance of the abstract `qb:ComponentProperty` class, which in turn has subclasses `qb:DimensionProperty`, `qb:AttributeProperty` and `qb:MeasureProperty`. Furthermore, such component properties can be typed additionally as `qb:CodedProperty` to attach (hierarchically organized) code lists to them. In addition to structure observations within datasets it is possible to organize observations within slices and groups using the respective concepts `qb:Slice` and `qb:ObservationGroup` and their related properties.

Exploring DataCubes using CubeViz. In order to hide the complexity of the RDF Data Cube vocabulary from users and to facilitate the browsing and exploration of DataCubes we developed the RDF DataCube browser *CubeViz*. CubeViz can be divided into two parts, both developed as an extension of OntoWiki:

1. A *Faceted data selection component*, which queries the structural part of a selected RDF graph containing DataCube resources.
2. A *Chart visualization component*, which queries observations (selected by the faceted selection component) and visualizes them with suitable charts.

CubeViz renders facets according to the DataCube vocabulary to select data on the first component, using SPARQL as the query language. Currently, the following facets are available:

1. Selection of a DataCube DataSet,
2. Selection of a DataCube Slice,
3. Selection of a specific measure and attribute (unit) property encoded in the respective DataCube dataset and
4. Selection of a set of dimension elements that are part of the dimensions encoded in the respective DataCube dataset.

As illustrated in Figure 6, all facets have a graphical user interface representation. Users have to click the icon next to the currently selected dataset to change the single choice selection of the first facet, which opens the respective dialogue. The slice facet, that offers a single choice selection, is only available if
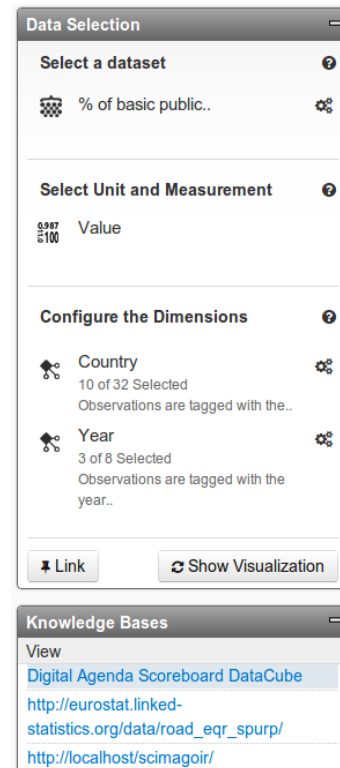

Fig. 6. Screenshot of CubeViz faceted data selection component

the selected RDF knowledge base contains materialized slices. All observations need to have at least one measurement, that can be selected, if more than one measurement is defined in the structure of the dataset. It is recommended to encode measurements with a respective unit definition using a resource of type `qb:AttributeProperty` but it is not mandatory. CubeViz only offers the single choice selection of attributes if at least one is defined. The last visible facet in Figure 6 offers abilities to select dimension elements of interest (slice and dice on-the-fly). Every dimension encoded in the dataset is listed together with their respective amount of available and selected dimension elements. After analyzing available chart implementations, CubeViz is aware of the maximum of realizable chart axis at one time. This amount of realizable chart axis is used to render the dialogues for selection of dimension elements – below this amount it is possible to have multiple choice selections, above this amount the user has to select exactly one element from the selection form.

A SPARQL query is generated to retrieve all matching observations, as a result of such a selection. Afterwards, the result set is analyzed to detect the amount of
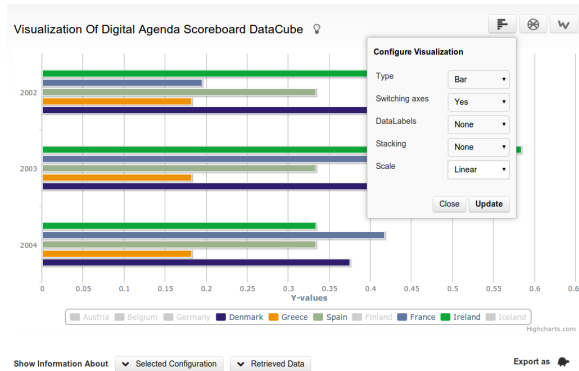
Fig. 7. Screenshot of the CubeViz chart visualization component

dimensions containing multiple elements and to select the charts that can be used to visualize the selected observation. As an outcome of the analysis, the first entry is selected from the chart list and the conditioned result set is assigned to it. Further configurations adjustable in CubeViz act on the visualization level. Users or domain experts are able to select different types of charts such as a bar chart, pie chart, line chart and polar chart that are offered depending on the selected amount of dimensions and its respective elements. CubeViz was developed focusing on easy adjustability of existing workflows and extensibility by libraries used to render the chart output. Currently, CubeViz renders charts on the client side using the JavaScript library *Highcharts*[13], a result of which is depicted in Figure 7. However, the integrated workflows allow an easy extension of existing charts or the addition of further chart libraries such as the *Google Charts*[14] or *Data Driven Documents*[15].

After rendering a chart, CubeViz offers chart-specific options, that can be used to adjust the output according to the users needs. For instance, in order to display widespread measurement values a logarithmic scale can be selected for improved visualization experience. Further integrated adjustment options are the chart subtype (offering combinations, e.g. polar/column chart) and the switch/inversion of the axis and dimensions. The set of adjustment options can be enhanced easily by adding/editing the specific option and/or the respective option values stored in the configuration file. After selection of observations and the adjustment of the chart the user is able to browse the metadata about the selected graph, the DataCube structure and the observations. Furthermore it is possible to share the results within a community using the permanent links, and to download the data in CSV or RDF-Turtle notation.

3.2.4. Content Management

One of the most frequently requested feature set during the last years of the project was the request for content management methods – more precisely, a methodology which allows for serving HTML representations of the resources of a specific knowledge base, in addition to RDF representations such as Turtle and RDF/XML.

The motivation for this requirement lies deeply in the workflow of special interest groups where people around a specific topic want to collect and share data about instances of their interest but do want also publish nice HTML pages based on these data. In this use case, OntoWiki acts as a content management backend where the data is managed while a frontend presents it to the non-editor user of the site. This distinction between frondend and backend user interface is typical for most content management systems[16] and the idea was to integrate a template engine as well as a method to provide public HTML representations, navigation menus, in-template queries and other template helpers.

With OntoWiki's site extension[17] a component was published which enables every OntoWiki instance to serve not only RDF but sophisitcated HTML representations of resources which are in the namespace of the instance. In the next paragraphs we introduce some aspects of the extension and outline its architecture.

URI design. The site extensions fosters a URI design based on well known file extension suffixes such as `ttl` for Turtle [8], `nt` for NTriples[18] and `rdf` for RDF/XML. This means that a resource `X`, such as `http://example.com/X` has specific information resources for different representations at `X.ttl`, `X.nt` and so on. A client which accesses `X` is then redirected to the most acceptable information resource based in its `Accept` HTTP request header field. In most cases, this is the HTML representation which is

---

[13]`http://www.highcharts.com/`
[14]`https://developers.google.com/chart/`
[15]`http://d3js.org/`

[16]Such as Drupal (`http://drupal.org`) and Wordpress (`http://wordpress.com`).
[17]`https://github.com/AKSW/site.ontowiki`
[18]`http://www.w3.org/2001/sw/RDFCore/ntriples/`

rendered by the site extension based on a template selection query[19].

RDFa Templates. In order to avoid non-semantic HTML output, the site extension provides multiple query helper to produce RDFa enhanced HTML snippets based on the data of the current resource. These helpers can be used as part of the templates itself[20] as well as as part of the content of the resources.

The implemented helper markup is used in the same way as well known wiki actions. The general syntax of a helper tag is: `{{tagname p1=value1 ... pX=valueX}}` where `p1 ... pX` where $pX \rightarrow valueX$ are key value pairs for parameter handover.

The following incomplete list of helper tags demonstrate their usage areas:

- The `query` helper uses a SPARQL `where` clause and renders the result set in a given sub template. This generic helper is used in a wide range of cases where dynamic lists of resources need to be created (list of current projects, lists of group members, ...).
- The `link` helper produces inline RDFa enabled HTML links to a specific resource. This helper is very useful when editing abstracts or running text of a resource description. Instead of using URIs for linking, the helper is able to select a resource by attribute search or full text search.
- The `img` helper consumes the description of an image resource and produces RDFa enabled HTML image code including caption, links and other characteristics.
- The `navigationlist` helper uses a cascading RDF sequence resource tree and builds a navigation menu out of the links. This allows site owners to manipulate their menus by changing attributes in the OntoWiki backend.

These are the most generic helpers which can be used in nearly all contexts. More specifically helpers generate HTML code based on the content of specific literal values (e.g. date, time and location coordinates). Each of the listed helpers outputs the literal content of the displayed resources in the requested language only. This is done by limiting the underlying queries to specific language tags and fall back languages. As a re-

sult, every OntoWiki hosted site can be translated just by providing different literals in the backend.

Extension vocabulary. In order to provide a method of manipulating and controlling the rendering internals as well as to provide a set of commonly used templates we created a site extension vocabulary which captures the available options. This includes properties to provide class and resource specific templates, to describe SPARQL queries and navigation structures.

As an example of a running site extension deployment we provide the content of our research group page `http://aksw.org`[21].

4. Authoring and Maintenance

To a large extent the overwhelming success of the World Wide Web was based on the ability of ordinary users to author content easily. In order to publish content on the WWW, users had to do little more than to annotate text files with few, easy-to-learn HTML tags. Unfortunately, on the semantic data web in general and more specific, on Linked Data authoring tools the situation is slightly more complicated. Users do not only have to learn a new syntax (such as Turtle, RDF/XML or RDFa), but also have to get acquainted with the RDF data model, ontology languages (such as RDFS, OWL) and a growing collection of connected RDF vocabularies for different use cases (such as FOAF, SKOS and SIOC). OntoWiki lowers this entrance barrier in authoring and maintaining semantic content by introducing editing interfaces which try to hide this complexity and still support all language features of the RDF.

4.1. RDFauthor

The *RDFauthor* approach is based on the idea of making arbitrary XHTML views with integrated RDFa annotations editable. *RDFa* [1] is the W3C Recommendation, which allows to combine human and machine-readable representations within a single HTML document. RDFauthor builds on RDFa by preserving provenance information in RDFa representations following the named-graph paradigm and by establishing a mapping from RDFa view representations to authoring widgets. On configurable events (such as

---

[19]Without the site extension enabled, OntoWiki would redirect to a resource view such as depicted in Figure 4.

[20]We use `phtml`-templates here, which is an easy to write HTML file with PHP annotations.

[21]The dataset can be downloaded at `http://aksw.org/model/export/?m=http%3A%2F%2Faksw.org%2F&f=turtle`, is annotated with VoID and listed at the `http://datahub.io`.

the clicking of a button or moving over a certain information fragment with the mouse) the widgets will be activated and allow the editing of all RDFa-annotated information on the Web page. While editing, the widgets can access background information sources on the Data Web in order to facilitate the reuse of identifiers or to encourage the interlinking of resources. Our resource editing widget, for example, suggests suitable, previously defined resources derived from calls to the Sindice Semantic Web index [40]. Once editing is completed, the changes are propagated to the underlying triple stores by means of the SPARQL/Update language. This allows for integration of RDF authoring widgets not only in OntoWiki itself but also inside of frontend websites generated with the site extension (refer to section 3.2.4).

In addition to that, RDFauthor is even not at all limited to editing semantic representations from a single source. An RDFa view made editable with RDFauthor can contain statements from a variety of sources, which can be edited simultaneously and in a wholly transparent manner for the user. Based on an extended RDFa markup supporting named graphs and SPARQL/Update endpoint information, simultaneous changes of several graphs from different sources will be dispatched to the respective SPARQL/Update endpoints. RDFauthor is implemented in JavaScript so that it works entirely on the browser side and can be used together with arbitrary Web application development techniques and is not tied to an integration in OntoWiki only.

RDFa enables the annotation of information encoded in HTML with RDF. This ability allows to extract a set of RDF triples from an RDFa-annotated HTML page. RDFauthor makes these triples editable, but in order to store changes persistently in the wiki, RDFauthor needs information about the data source (i. e. SPARQL and SPARQL/Update endpoint) regarding the named RDF graph from which the triples were obtained or where they have to be updated. In order to make this information available, we have defined a slight extension of the RDFa annotations.

To represent information about the information source, we follow the named graphs approach [12]. We created a small schema[22] to represent attributes and relations for the following purposes:

– In order to link certain RDFa annotations on the page to the respective querying/update services, namely SPARQL/Update and SPARQL endpoints, we propose the use of the `link` HTML tag with an `about`-attribute to identify the named graph, a `rel`-attribute with the value `update:updateEndpoint` and the HTML `href` attribute with the URL of the respective SPARQL/Update endpoint. Another option to declare graph metadata is the use of empty `span`- or `div`-elements together with the RDFa attributes inside the body of the page.
– For declaring which statements belong to which named graph, we propose the use of the `update:from` attribute with the named graph as attribute value to which all nested RDFa annotations should belong. The `update:from` attribute and the additional RDFa processing rules are inspired by [22]. The use of named graphs is optional and only required, if triples from multiple sources should be made editable.

OntoWiki includes graph metadata in every single list and resource view which is generated. In addition to that, the site extension can include this data based on template helpers. This allows for triggering the generation of authoring widgets through a number of different trigger events. These events can be grouped into element-based events or page-wide events. In particular, the following triggers are supported:

– Clicking on an edit button next to an element containing the object of a statement,
– moving the pointer and hovering above an object element,
– an application-specified custom trigger such as the "Edit Properties" in OntoWiki,
– a bookmarklet which loads all RDFauthor components and runs all widgets at once,
– the universal edit button[23] which runs the same bookmarklet by using a browser button.

Upon user interaction or a programmatic trigger, RDFauthor starts processing the current page by extracting all RDF triples and placing them in an *rdf-Query databank*; one for each named graph. Triples that describe the named graphs in the page by using the update vocabulary are excluded from editing. If no update information has been defined for a graph, it is

---

[22]The RDFauthor vocabulary namespace is `http://ns.aksw.org/update/`. We use the prefix `update` for this namespace throughout this paper.

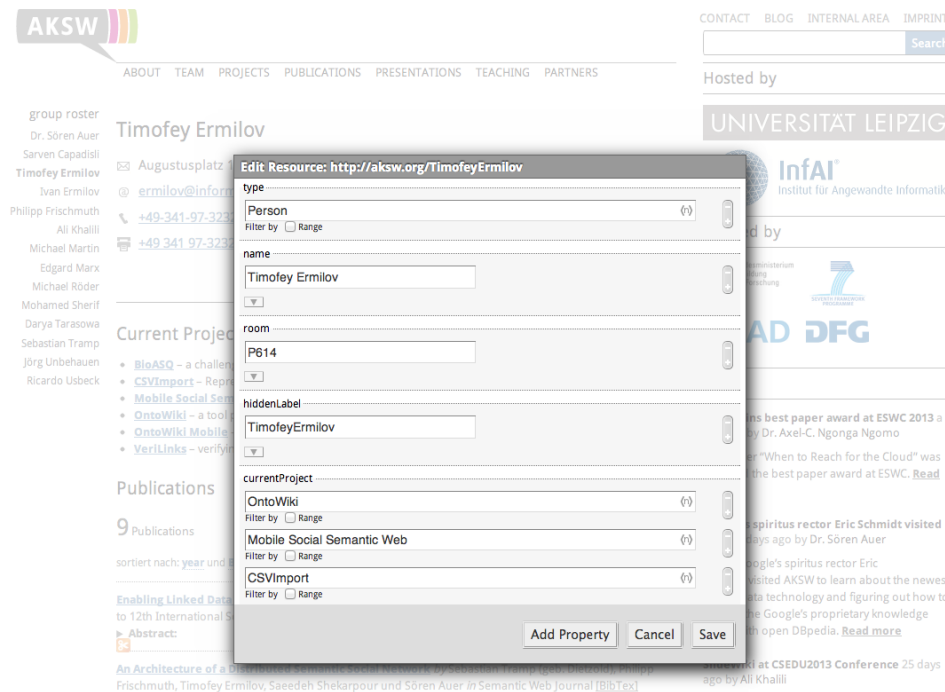[23]`http://universaleditbutton.org`

Fig. 8. RDFauthor Widget Example: This editing view was triggered by the universal edit button integrated in the underlying profile page and is added on top of the complete site as a modal view. After submitting the changes back to the Wiki by using the Save button, SPARQL update requests communicate the added or deleted triples.

considered non-editable, hence no form elements are created for the triples it contains.

If the named graph from which the statement originates is linked to a SPARQL endpoint RDFauthor tries to retrieve property metadata from the endpoint by querying the `rdf:type` and `rdfs:range` of the property description. Based on this information, a corresponding editing widget is selected. All selected widgets are combined into an edit view and are displayed to the user. Depending on the type of trigger, this edit view can be rendered in two ways: overlay window or integration view. Figure 8 depicts an example of an RDFauthor generated form in an overlay window.

When the user finishes the editing process, all widgets involved are asked to update the respective named graph with their changes. The difference between the original and modified graphs are calculated (i. e. added statements, removed statements), yielding a diff graph. The associated store to each graph is then updated with the respective diff graph by means of SPARQL/Update [17] operations. By explicitly listing all inserted or deleted triples using `INSERT DATA` and `DELETE DATA` syntax, sophisticated SPARQL/Update support is not required. In addition, RDFauthor can cope with several access control scenarios. It,

therefore, evaluates the server's response to SPARQL/Update requests. For instance, in the case of an HTTP 401 (unauthorized) or 403 (forbidden) status code, a login form is displayed.

In addition to modifying the triple content of a page, it is possible to add new statements. This can happen either based on existing triples used as templates or by adding entirely new statements. If existing triples are used as templates, three cases can be distinguished:

– Creating a new statement that shares subject and property with an existing statement. Our approach supports this case via a small button beside each statement.
– Creating a new statement that shares the subject with an existing statement. At the end of a subject description a small button is shown which lets the user add a new statement to the subject's description.
– Creating a new resource using an existing resource as a template. Widgets for all properties found on the template resource are available on the new resource.

Adding new properties to an existing resource is accomplished in two steps. First, the user chooses a prop-
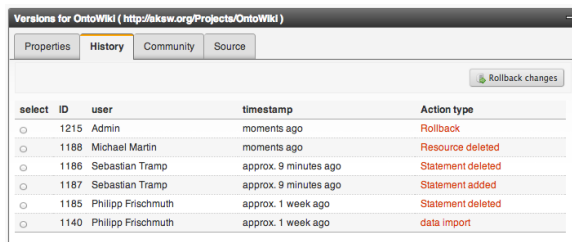
erty which she wants to use. She types a name or description fragment into the search input field of the property widget and RDFauthor searches for properties in the referenced SPARQL endpoint of the given named graph. Subsequently, the corresponding widget is selected from the existing widget library.

4.2. Versioning

Versioning of wiki content is, according to Leuf and Cunning [25], a fundamental part of wiki systems. To roll back a previous change helps users of a wiki to fix mistakes as well as to overcome their fear to damage the whole system with their user input.

The OntoWiki versioning system is implemented based on added and deleted triples in a change set. Since change sets can be pushed over different channels, such as the SPARQL endpoint or the JSON/RPC interface, the versioning system is integrated deeply into the triple store access functionality of the Erfurt API.

Fig. 9. Timeline of an example resource which starts with a data import, followed by some small changes and concluded by an accidental deletion and the rollback of this action.

This means that every update request will cause a calculation of added and deleted statements in order to be able to roll back this change. Multiple update requests can be handled as one single transaction of changes. In addition to that, each change set is tagged with the acting agent, the time stamp and an optional transaction name for user convenience. These transaction names can be used as IDs for a timeline view as depicted in figure 9.

4.3. Import and Triplification of Resources

An important feature which helps in daily data activities is to import different small and big chunks of data from different sources and optionally transform foreign non-RDF resources to RDF. First of all, OntoWiki is able to fetch and integrate data via linked data.

This is important e.g. for gathering property and class descriptions from the schema namespace document.

OntoWiki is also able to parse content of RDFa pages and can be extended with other data gathering plugins in order to support non-RDF import of other structured data (e.g. EXIF data).

In addition to this, OntoWiki is able to use foreign SPARQL endpoints to provide read-only knowledge bases inside of a local deployment. Such endpoints can be provided by RDB2RDF mapping tools such as Sparqlify [41] in order to integrate data from relational databases.

4.4. Evolution of Datasets

In order to allow more sophisticated manipulation activities we integrated an approach for writing, using and managing evolution patterns. The *EvoPat* [35] approach is based on the definition of basic evolution patterns, which are represented declaratively and can capture simple evolution and refactoring operations on both data and schema levels. For more advanced and domain-specific evolution and refactorings, several simple evolution patterns can be combined into a compound one. In [35] we performed a comprehensive survey of possible evolution patterns with a combinatorial analysis of all possible before/after combinations, resulting in an extensive catalog of usable evolution patterns. The execution of an evolution pattern on a knowledge base results in requesting multiple changes as one single transaction. If the result is not as intended, a roll back can revert these changes easily.

5. Machine Consumption

Up to this section we presented a variety of discovery and exploration as well as authoring and maintenance methods that are provided by OntoWiki. Those methods are to a large extend targeted at human users. In this section we present additional interfaces that OntoWiki provides in regard to machine consumption.

5.1. SPARQL Endpoint

OntoWiki internally relies on a SPARQL capable triple store. Hence, this functionality can also be made available externally adhering to the SPARQL protocol [16]. The SPARQL query service of OntoWiki by default listens at `<OntoWikiRoot>/sparql` and requires only a single mandatory parameter:

`query=EncodedQuery`. When provided, the optional parameters

- `default-graph-uri` and
- `named-graph-uri`

are also evaluated. By default, the SPARQL endpoint responds with a

- `application/sparql-results+xml` content-type for `ASK` and `SELECT` queries, as well as with
- a `application/rdf+xml` content-type for `CONSTRUCT` queries.

Instead of making all data contained in the underlying RDF store available, OntoWiki applies the same access control rules for the endpoint that are evaluated within the tool itself. Only those graphs visible to the user can be queried with SPARQL.

5.1.1. Linked Data Endpoint

Without further configuration, OntoWiki automatically publishes all resources according to the Linked Data principles [9] as long as they use the same namespace as the OntoWiki deployment. Since the tool is generic and works with arbitrary RDF data, this is not always the case. But when it comes to authoring data with OntoWiki, resources are automatically created using the correct namespace and thus are Linked Data capable.

Another requirement is that resource URIs must not collide with URLs used for OntoWiki functionality. Since the number of such URLs is very small and the number of possible URIs within a namespace in comparison is very large, this will rarely lead to problems.

For the publication process OntoWiki utilizes the 303 approach, which yields in multiple URIs. If a client requests a Linked Data resource, it will never get the result in the format it requested (determined via content negotiation) directly. Instead a 303 HTTP response will be send back to the client with an appropriate `Location` header field. A human consumer usually uses a web browser and thus would request a HTML representation of the resource. Hence OntoWiki would return the URL of the generic resource view in its 303 response. Another tool would more likely request any of the RDF serialization formats. In this case OntoWiki would return a URL that leads to a suitable export in that format, iff OntoWiki supports it.

5.1.2. Other Interfaces

In addition to the above introduced interfaces for machine consumption, OntoWiki has further interfaces.

A JSON-RPC[24] gateway exposes a wide range of functionality to be used by other applications, such as for example owcli, the *OntoWiki Command Line Interface*[25].

Furthermore OntoWiki provides support for the Semantic Pingback protocol [38] in order to improve the interlinking within the (Data) Web.

Support for the pubsubhubbub[26] protocol permits, that other applications can subscribe to feeds provided by OntoWiki (e.g. changes on resources) and retrieve updates immediately.

6. Experiences with OntoWiki

6.1. Enterprise Data Integration

Nowadays, almost every large enterprise uses taxonomies to provide a shared linguistic model aiming at structuring the large quantities of documents, emails, product descriptions, enterprise directives, etc. which are produced on a daily basis. However, those taxonomies often are stored in proprietary formats as well as maintained and controlled by certain departments. This creates a considerable barrier for the use and especially reuse of such data.

In an industry project with a large enterprise, we employed OntoWiki to improve this situation by converting existing dictionaries containing term definitions in multiple languages into RDF. We utilized the standardized and popular SKOS vocabulary for this purpose and published all term definitions via the Linked Data principles.

As a first result all terms (that were previously spread over multiple dictionary files) were available in a unified knowledge base and users were able to comfortably browse the taxonomy using OntoWiki.

To showcase the benefits of making the enterprise taxonomy available as RDF (especially the reusability in other scenarios), we converted another data source into RDF, which contains structured information about the products the company offers (cars). We linked those products to terms in the taxonomy and built a

---

[24]`http://json-rpc.org/`
[25]`https://github.com/AKSW/owcli`
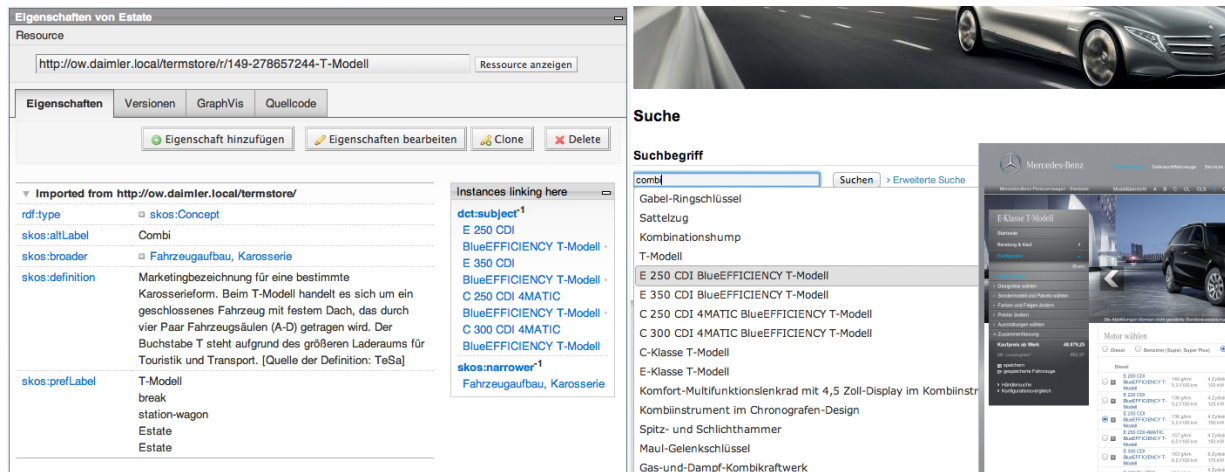[26]`https://code.google.com/p/pubsubhubbub/`

Fig. 10. The left side shows OntoWiki, which displays the definition of the term *T-Modell* from the taxonomy and other resources that link to this term. The right side shows a search application after searching for *combi*, which employs the term metadata as well as the links to this very concept for finding and suggesting relevant content.

custom search service, which is depicted in Figure 10. The screenshot on the left side shows OntoWiki, which displays the definition of the term *T-Modell* contained in the taxonomy graph along with some additional information. The location bar on the top of the screen displays the URI used for this very concept, which other resources can link to. It is also possible to directly de-reference this identifier and obtain the description for this resource in a machine-readable format. The properties table for this term shows:

- the type of this resource (`skos:Concept`),
- a link to a concept that is broader (hierarchical order),
- a textual description of the meaning of this term,
- preferred labels for this term in different languages as well as
- an alternative label *Combi*.

Additionally, a small box on the right side of the OntoWiki screen shows other resources that link to this term. As one can imagine, the broader concept from above also contains a link to this term stating that it is a narrower term (`skos:narrower`). More interestingly the other links show that certain car models link to this concept. This circumstance is used in the search application, which is shown on the right side of Figure 10. This screenshot shows a simple prototype application with a search field. When a user types the keyword *combi*, the knowledge base is used to obtain the fact, that this search term is a synonym for the concept *T-Modell*. Once this is done, all linked car models

are retrieved and shown to the user. The depicted scenario is a good example of an application of a taxonomy outside the scope of the originally intended use.

6.2. Open Data Publication and Visualization

Two main advantages of the Linked Data paradigm are standardized techniques and workflows for publishing and consuming structured data. It is possible to categorize applications according to their focus [26]. On the one hand, applications can facilitate re-use of their data through a *Linked Data endpoint* publishing dereferencable RDF resources and, in addition, a *SPARQL endpoint* to make the data available through custom queries. On the other hand, applications that consume Linked Data (e.g. to create custom views on the data) contain access interfaces for querying data via SPARQL and to receive RDF resources using dereferencable URIs. In the following we describe two use cases wherein customized versions of OntoWiki were deployed. In addition to capitalizing on the advantages of Linked Data for developers, both use cases focus on human consumption interfaces.

Financial Transparency Data of the EC. The *Financial Transparency System* (FTS) of the European Commission contains information about commitments granted and supported by the European Union starting from 2007. It allows users to get an overview on EU funding, including information on beneficiaries as well as the amount and type of expenditure and information on the responsible EU department. The original dataset

is freely available on the European Commission website, where users can query the data using an HTML form and download it in CSV and most recently XML format [27].

In order to publish this dataset (1) which can be reused, (2) which can allow more complex and interesting queries, we analyzed the original dataset, developed a vocabulary and converted the data into RDF [27]. Afterwards, we geocoded the spatial resources and interlinked further resources with other datasets from the Linked Data Web to increase its added value. In addition to providing the RDF dataset as a dump for download, we also published the data according to the Linked Data principles. All URIs minted during the transformation process are within a dedicated FTS namespace[28]. After publishing FTS data, we added related meta information (partially supported by OntoWiki's VoID plugin[29]) to the `owl:Ontology` node of the FTS graph such as authors, contributors, the last modification date (currently: October, 10, 2012) and the license (Creative Commence License CC-BY-3.0).

OntoWiki is used to publish the FTS dataset as Linked Data and to proceed FTS specific optimizations. Since the instance of OntoWiki for the FTS is not to be used as a data acquisition Wiki, but instead only as a publishing and consumption interface for machines and humans, we disabled all authentication and editing interfaces. A static landing page[30] was created to present details about the dataset such as content descriptions, original sources, examples, dumps to download and contact information.

FTS was published in September 2012 and approximately 1.000 unique users visited the site per month. Furthermore, we successfully tested the SPARQL and Linked Data endpoint of FTS using *Facete*[31], an advancement of the *LinkedGeodataBrowser*[32], that facilitates browsing of geocoded RDF resources and related resource chains as illustrated in Figure 11.

European Data Portal and statistical data discovery. The European Data Portal was launched in 2012 (cf. subsubsection 3.2.3). It is a data catalog based on *CKAN*[33] publishing meta information about publicly available datasets. Most of them deal with statistical data and can be downloaded or browsed with specially created web applications. One of the tools, that will be integrated and listed on the European Data Portal is a customized version of OntoWiki in combination with CubeViz. Since the responsible department hosting the portal does not own and maintain most of the listed datasets, CubeViz can not be setup to receive data from a local triple store. Therefore, CubeViz will act as a human consumption interface to provide easy access and discovery of statistical data to users. As a result, we deployed OntoWiki with a customized theme adhering to the EC Open Data Portal style guide and a trimmed set of necessary extensions such as the CubeViz component.

Further extensions that have been developed for the portal are *translations* to support multi-linguality, *static links*[34] and *page*[35] to publish explanations about the usage of the tool and the *defaultmodel* extension that is necessary to start the tool without knowledge about available datasets. We deployed a specially configured SPARQL backend for the use of OntoWiki without a local triple store. Maintainers only have to configure a specific SPARQL endpoint and the set of graph IRIs used to identify therein hosted datasets in order to use this backend. If the user selects a dataset containing a well-formed DataCube (according to the integrity constraints published by the W3C[36]), the user is able to browse the dataset as illustrated in Figure 12.

6.3. Prospographical database Catalogus Professorum Lipsiensium

The World Wide Web, as an ubiquitous medium for publication and exchange, has already significantly influenced the way historians work: the online availability of catalogs and bibliographies allows to efficiently search for content relevant for a certain investigation; the increasing digitization of works from historical archives and libraries, in addition, enables historians to directly access historical sources remotely. The capabilities of the web as a medium for collaboration, however, is part of many initiatives in the field of digital humanities. Many, historical questions can only be answered by combining information from different sources, from different researchers and organizations.

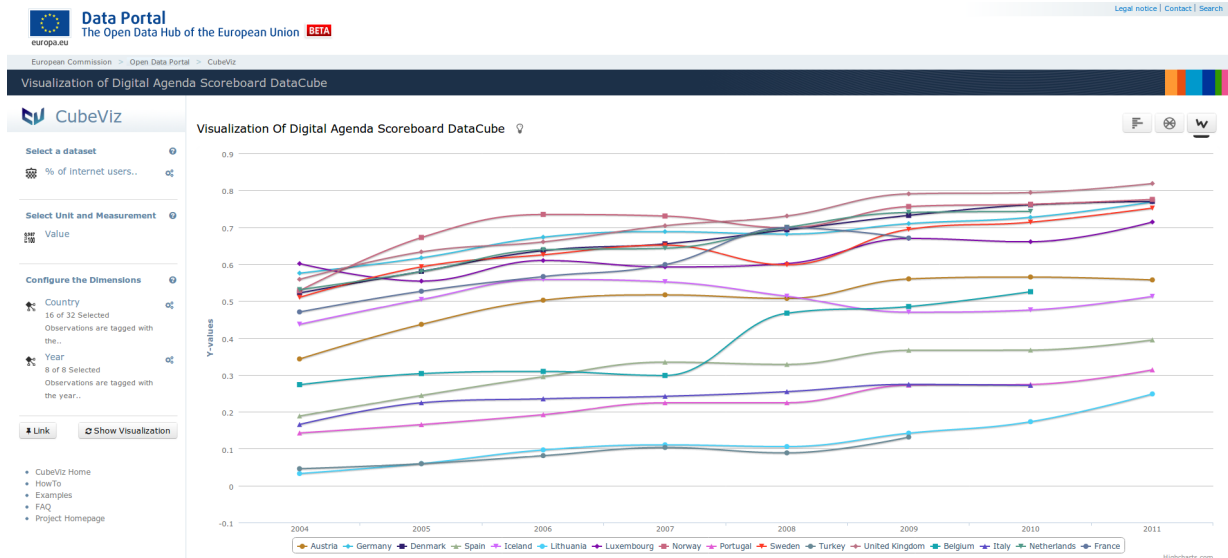Fig. 11. Screenshot of Facete browsing FTS.

Fig. 12. Screenshot of CubeViz as part of EC's Open Data Portal.

If the original sources are analyzed, the derived information is often much richer, than can be captured by simple keyword indexing. These factors pave the way for the successful application of knowledge engineering techniques in historical research communities.

Fig. 13. Public interface of CPL

With the Catalogus Professorum Lipsiensium (CPL) we developed an adaptive, semantics-based knowledge engineering application based on OntoWiki for prosopographical knowledge [34]. In prosopographical research, historians analyze common characteristics of historical groups by studying statistically relevant quantities of individual biographies. Untraceable periods of biographies can be determined on the basis of such accomplished analyses in combination with statistically examinations as well as patterns of relationships between individuals and their activities.

Researchers from the Historical Seminar at University of Leipzig aimed at creating a prosopographical knowledge base about the life and work of professors in the 600 years history of University of Leipzig ranging from the year 1409 till 2009 – the *Catalogus Professorum Lipsiensium*. In order to enable historians to collect, structure and publish this prosopographical knowledge an vocabulary was developed and incrementally refined over a period of three years. The *Catalogus Professorum Model*[37] (CPM) comprises several ontologies and vocabularies for structuring the prosopographical information. The model consist of concepts like Person, Body or Period of Life, that are individually modeled and interlinked.

The community of historians working on the project was enabled to add information to the knowledge base using an adapted version of the semantic Data Wiki OntoWiki. For the general public, a simplified user interface[38] is dynamically generated based on the content of the knowledge base (cf. Figure 13, Figure 14). For access and exploration of the knowledge base by other historians a number of access interfaces was developed and deployed, such as a graphical SPARQL query builder, a relationship finder [20] and plain RDF and Linked Data interfaces. As a result, a group of 10 historians supported by a much larger group of volunteers and external contributors collected information about 2.000 professors, 15.000 associated periods of life, 400 institutions and many more related entities.

The system architecture of CPL combines different applications, which interact using standardized interfaces as illustrated in Figure 14. It is divided in a public and a protected zone due to technical constraints and in order to prevent security problems. The semantic Data Wiki OntoWiki located in the protected layer [39] uses the *Catalogus Professorum Model* for structuring the prosopographical information. The project team, consisting of historians supported by knowledge engineers and Semantic Web experts, is working collaboratively and spatially distributed (e.g. in archives or libraries) to collect, structure and validate information about persons and institutions relevant to this knowledge domain. The resulting knowledge base is accessible only by the project team and is backed-up nightly.

Using two configurable tools the knowledge base is exported in order to make it accessible for the public. Domain experts, i.e. historians, are able to interact with CPL via an experimental version[40] of OntoWiki. The version of the catalog available there is synchronized using the tool OCPY[41] (**O**ntology **Co**PY), that exports data from the protected OntoWiki installation, transforms the exported data considering any linked knowledge bases and imports the changed data into this experimental installation. This experimental deployment in particular offers new functionality of OntoWiki for testing purposes.

---

[37]Catalogus Professorum Model: http://catalogus-professorum.org/model/

[38]Available at: http://www.uni-leipzig.de/unigeschichte/professorenkatalog/

[39]http://professoren.ontowiki.net [restricted access]; OntoWiki-Version 0.85

[40]http://catalogus-professorum.org/

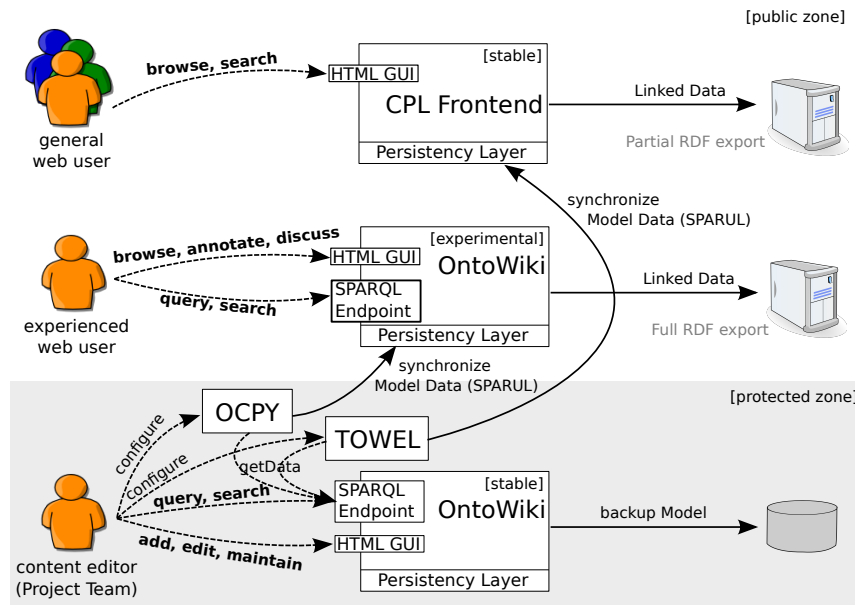[41]http://catalogus-professorum.org/tools/ocpy/

Fig. 14. Architectural overview about the CPL project platforms

As users of this web interface are interested in using the newest technologies it is strictly separated from the main and stable layer. The knowledge base that is provided there is updated at least once per month. The script OCPY exports, transforms and imports the knowledge base according to the latest version of the vocabulary and takes care of additional linked knowledge bases that provide e.g. geographical informations using the *Spatial Hierarchy Ontology*[42].

The benefits of the developed knowledge engineering platform for historians are twofold: Firstly, the collaboration between the participating historians has significantly improved: The ontological structuring helped to quickly establish a common understanding of the domain. Collaborators within the project, peers in the historic community as well as the general public were enabled to directly observe the progress, thus facilitating peer-review, feedback and giving direct benefits to the contributors. Secondly, the ontological representation of the knowledge facilitated original historical investigations, such as historical social network analysis, professor appointment analysis (e.g. with regard to the influence of cousin-hood or political influence) or the relation between religion and university.

The use of the developed model and knowledge engineering techniques is easily transferable to other prosopographical research projects and with adaptations to the ontology model to other historical research in general. In the long term, the use of collaborative knowledge engineering in historian research communities can facilitate the transition from largely individual-driven research, where one historian investigates a certain research question solitarily, to more community-oriented research, where many participants contribute pieces of information in order to enlighten a larger research question. Also, this will improve the reusability of the results of historic research, since knowledge represented in structured ways can be used for previously not anticipated research questions.

7. Conclusions and Future Work

In this article, we provided a comprehensive presentation of the concepts, technical architecture and interfaces of the semantic Data Wiki OntoWiki. We showcased three complementary use cases in different semantic knowledge engineering and management domains thus proving OntoWiki's versatility. During the development and application of OntoWiki we encountered a number of challenges. The main challenge from our experience is the *balancing of scalability, usability and functionality*. Each of these three areas is of crucial importance for certain applications and use cases but they mutually impact each other. For example, adding

---

[42]http://ns.aksw.org/spatialHierarchy/

more features or improving usability might jeopardize scalability. Also, adding more features can overload the user interface and complicate usability. As a result, even though OntoWiki in its core is generic and domain agnostic, it is of paramount importance to make the system flexible to adapt to new usage scenarios. For that purpose, we integrated a number of measures, such as a modular, layered MVC architecture, plug-in and extension interfaces, high-configurability through system models, binding of visualization components to SPARQL query result sets, etc.

Although the resulting system already fulfills the requirements of industrial strength applications, much more work has to be done in order to further reduce the entrance barrier for the development of Semantic Web applications. We plan to continue our work with regard to optimizing the balancing within the scalability, usability and functionality triangle. We plan to further increase the flexibility by better leveraging the relationships between vocabularies, configuration knowledge and background knowledge available on the Data Web. A particularly promising avenue of research and development is also the tighter integration of OntoWiki with other stages of the Linked Data life-cycle [5]. A first step in this direction was performed with the release of the LOD2 Stack [2], which includes in addition to OntoWiki for the authoring, visualization and exploration stages a number of other tools for data linking, quality improvement, enrichment, evolution and visualization. By continuing to work in this direction, we hope that OntoWiki will help to make the Web a place, where structured data publishing and consumption can be performed in a distributed, decentralized, semantically heterogeneous and still collaborative way thus facilitating to the long tail of information domains and usage scenarios.

Acknowledgments

References

[1] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and Processing. W3C Recommendation, World Wide Web Consortium (W3C), Oct. 2008. http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014.

[2] S. Auer, L. Bühmann, C. Dirschl, O. Erling, M. Hausenblas, R. Isele, J. Lehmann, M. Martin, P. N. Mendes, B. van Nuffelen, C. Stadler, S. Tramp, and H. Williams. Managing the Life-Cycle of Linked Data with the LOD2 Stack. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, editors, *The Semantic Web - – ISWC 2012, 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II*, volume 7650 of *Lecture Notes in Computer Science*, pages 1–16, Berlin / Heidelberg, 2012. Springer.

[3] S. Auer, J. Demter, M. Martin, and J. Lehmann. LOD-Stats — An Extensible Framework for High-Performance Dataset Analytics. In A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Acquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, editors, *Knowledge Engineering and Knowledge Management, 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, volume 7603 of *Lecture Notes in Computer Science*, pages 353–362, Berlin / Heidelberg, 2012. Springer.

[4] S. Auer, S. Dietzold, and T. Riechert. OntoWiki – A Tool for Social, Semantic Collaboration. In I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. M. Aroyo, editors, *The Semantic Web – ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006. Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749, Berlin / Heidelberg, 2006. Springer.

[5] S. Auer, J. Lehmann, A.-C. N. Ngomo, and A. Zaveri. Introduction to Linked Data and Its Lifecycle on the Web. In S. Rudolph, G. Gottlob, I. Horrocks, and F. van Harmelen, editors, *Reasoning Web. Semantic Technologies for Intelligent Data Access, 9th International Summer School 2013, Mannheim, Germany, July 30 – August 2, 2013. Proceedings*, volume 8067 of *Lecture Notes in Computer Science*, pages 1–90, Berlin / Heidelberg, 2013. Springer.

[6] D. Aumüller. Semantic Authoring and Retrieval within a Wiki (WikSAR). In Demo Session at the Second European Semantic Web Conference (ESWC2005), May 2005, Heraklion, Crete, Greece. Available at http://wiksar.sf.net, 2005.

[7] D. Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, World Wide Web Consortium (W3C), Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[8] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. RDF 1.1 Turtle – Terse RDF Triple Language. W3C Recommendation, World Wide Web Consortium (W3C), Feb. 2014. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[9] T. Berners-Lee. Linked Data. W3C Design Issues, World Wide Web Consortium (W3C), July 2006. http://www.w3.org/DesignIssues/LinkedData.html.

[10] J. G. Breslin, A. Harth, U. Bojars, and S. Decker. Towards Semantically-Interlinked Online Communities. In A. Gómez-

Pérez and J. Euzenat, editors, *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29–June 1, 2005. Proceedings*, volume 3532 of *Lecture Notes in Computer Science*, pages 500–514, Berlin / Heidelberg, 2005. Springer.

[11] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, World Wide Web Consortium (W3C), Feb. 2004. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[12] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named Graphs, Provenance and Trust. In A. E. Southern and T. Hagino, editors, *Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM.

[13] R. Cyganiak, S. Field, A. Gregory, W. Halb, and J. Tennison. Semantic Statistics: Bringing Together SDMX and SCOVO. In C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, editors, *LDOW-2010, Linked Data on the Web 2010, Proceedings of the WWW2010 Workshop on Linked Data on the Web, Raleigh, USA, April 27, 2010.*, volume 628 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. `http://ceur-ws.org/Vol-628/`.

[14] R. Cyganiak, D. Reynolds, and J. Tennison. The RDF Data Cube Vocabulary. W3C Recommendation, World Wide Web Consortium (W3C), Jan. 2014. `http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/`.

[15] O. Erling and I. Mikhailov. RDF Support in the Virtuoso DBMS. In S. Auer, C. Bizer, C. Müller, and A. v. Zhadanova, editors, *The Social Semantic Web 2007: Proceedings of the First Conference on Social Semantic Web (CSSW), September 26-28, 2007, Leipzig, Germany*, volume 113 of *GI-Edition – Lecture Notes in Informatics (LNI)*, pages 59–68, Bonn, 2007. Gesellschaft für Informatik (GI).

[16] L. Feigenbaum, G. T. Williams, K. G. Clark, and E. Torres. SPARQL 1.1 Protocol. W3C Recommendation, World Wide Web Consortium (W3C), Mar. 2013. `http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/`.

[17] P. Gearon, A. Passant, and A. Polleres. SPARQL 1.1 Update. W3C Recommendation, World Wide Web Consortium (W3C), Mar. 2013. `http://www.w3.org/TR/2013/REC-sparql11-update-20130321/`.

[18] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, World Wide Web Consortium (W3C), Mar. 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[19] M. Hausenblas, W. Halb, Y. Raimond, L. Feigenbaum, and D. Ayers. SCOVO: Using Statistics on the Web of Data. In L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, M. S. Eyal Oren, and E. Simperl, editors, *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009 Heraklion, Crete, Greece, May 31–June 4, 2009 Proceedings*, volume 5554 of *Lecture Notes in Computer Science*, pages 708–722, Berlin / Heidelberg, 2009. Springer.

[20] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing Relationships in RDF Knowledge Bases. In T.-S. Chua, Y. Kompatsiaris, B. Mérialdo, W. Haas, G. Thallinger, and W. Bailer, editors, *Semantic Multimedia, 4th International Conference on Semantic and Digital Media Technologies, SAMT 2009 Graz, Austria, December 2-4, 2009 Pro-*

ceedings, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187, Berlin / Heidelberg, 2009. Springer.

[21] N. Heino, S. Dietzold, M. Martin, and S. Auer. Developing Semantic Web Applications with the OntoWiki Framework. In T. Pellegrini, S. Auer, K. Tochtermann, and S. Schaffert, editors, *Networked Knowledge – Networked Media, Integrating Knowledge Management, New Media Technologies and Semantic Systems*, volume 221 of *Studies in Computational Intelligence*, pages 61–77. Springer, Berlin / Heidelberg, 2009.

[22] T. Inkster and K. Kjernsmo. Named Graphs in RDFa (RDFa Quads). Buzzword.org.uk Draft, Buzzword.org.uk, Jan. 2009. `http://buzzword.org.uk/2009/rdfa4/spec-20090120`.

[23] ISO. Statistical data and metadata exchange (SDMX). Standard No. ISO/TS 17369:2005, ISO, 2005.

[24] M. Krötzsch, D. Vrandečić, M. Völkel, H. Haller, and R. Studer. Semantic Wikipedia. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier*, vol. 5, issue 4:251–261, Dec. 2007.

[25] B. Leuf and W. Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Professional, Apr. 2001.

[26] M. Martin and S. Auer. Categorisation of Semantic Web Applications. In *Proceedings of the 4th International Conference on Advances in Semantic Processing (SEMAPRO2010), October 25-30, Florence, Italy*, Oct. 2010.

[27] M. Martin, C. Stadler, P. Frischmuth, and J. Lehmann. Increasing the financial transparency of european commission project funding. *Semantic Web Journal, Special Call for Linked Dataset descriptions, IOS Press*, 5, 2/2014:157–164, 2014.

[28] M. Martin, J. Unbehauen, and S. Auer. Improving the Performance of Semantic Web Applications with SPARQL Query Caching. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 – June 3, 2010, Proceedings, Part II*, volume 6089 of *Lecture Notes in Computer Science*, pages 304–318, Berlin / Heidelberg, 2010. Springer.

[29] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, World Wide Web Consortium (W3C), Feb. 2004. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[30] A. Miles and S. Bechhofer. SKOS Simple Knowledge Organization System Reference, Aug. 2009. `http://www.w3.org/TR/2009/REC-skos-reference-20090818/`.

[31] U. Nations. Guidelines for Statistical Metadata on the Internet. Technical report, United Nations Statistical Commission and Economic Commission for Europe (UNECE), Geneva, 2000. `http://www.unece.org/fileadmin/DAM/stats/publications/metadata.pdf`.

[32] OECD. Management of Statistical Metadata at the OECD. Technical report, Organisation for Economic Co-operation and Development (OECD), Sept. 2006. `http://www.oecd.org/std/33869551.pdf`.

[33] E. Oren. SemperWiki: a semantic personal Wiki. In S. Decker, J. Park, D. Quan, and L. Sauermann, editors, *Semantic Desktop Workshop, Proceedings of the ISWC 2005 Workshop on The Semantic Desktop – Next Generation Information Management & Collaboration Infrastructure. Galway, Ireland, November 6, 2005*, volume 175 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005. `http://ceur-ws.org/Vol-175/`.

[34] T. Riechert, U. Morgenstern, S. Auer, S. Tramp, and M. Martin. Knowledge Engineering for Historians on the Example of the Catalogus Professorum Lipsiensis. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, editors, *The Semantic Web — ISWC 2010, 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part II*, volume 6497 of *Lecture Notes in Computer Science*, pages 225–240, Berlin / Heidelberg, 2010. Springer.

[35] C. Rieß, N. Heino, S. Tramp, and S. Auer. EvoPat -– Pattern-Based Evolution and Refactoring of RDF Knowledge Bases. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, editors, *The Semantic Web — ISWC 2010, 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes in Computer Science*, pages 647–662, Berlin / Heidelberg, 2010. Springer.

[36] S. Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *Proceedings of the 1st International Workshop on Semantic Technologies in Collaborative Applications (STICA), June 26-28, 2006, Manchester, U.K.*, 2006.

[37] A. Souzis. Building a Semantic Wiki. *IEEE Intelligent Systems*, 20(5):87–91, 2005.

[38] S. Tramp, P. Frischmuth, T. Ermilov, and S. Auer. Weaving a Social Data Web with Semantic Pingback. In P. Cimiano and H. S. Pinto, editors, *Knowledge Engineering and Management by the Masses, 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings*, volume 6317 of *Lecture Notes in Computer Science*, pages 135–149, Berlin / Heidelberg, 2010. Springer.

[39] S. Tramp, N. Heino, S. Auer, and P. Frischmuth. RDFauthor: Employing RDFa for Collaborative Knowledge Engineering. In P. Cimiano and H. S. Pinto, editors, *Knowledge Engineering and Management by the Masses, 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings*, volume 6317 of *Lecture Notes in Computer Science*, pages 90–104, Berlin / Heidelberg, 2010. Springer.

[40] G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the Open Linked Data. In K. Aberer, K.-S. C. qnd Natasha Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*, volume 4825 of *Lecture Notes in Computer Science*, pages 552–565, Berlin / Heidelberg, 2007. Springer.

[41] J. Unbehauen, C. Stadler, and S. Auer. Accessing Relational Data on the Web with SparqlMap. In H. Takeda, Y. Qu, R. Mizoguchi, and Y. Kitamura, editors, *Semantic Technology, Second Joint International Conference, JIST 2012, Nara, Japan, December 2-4, 2012. Proceedings*, volume 7774 of *Lecture Notes in Computer Science*, pages 65–80, Berlin / Heidelberg, 2013. Springer.