# Chapter 6
# Semantic Data Interfaces for the Masses

**Abstract** Extensive research has been done on models, theoretical approaches and development of tools to infer new information from data and ontologies. However, the Semantic Web will only be fully integrated with the everyday-Web only when it will be easily available and fully accessible to every Web users and not only to Semantic Web practitioners. The Semantic Publishing community urgently needs specific tools to assist non-experts in dealing with and publishing semantic data. In this chapter, I describe influential work and my personal contribution to this area. In addition, I discuss possible next steps to extend existing legislative editor by means of Semantic Web technologies, and highlight the key characteristics of semantically-enhanced legislative editors of the future.

Any policy that guarantees the wide adoption of Semantic Web technologies, such as those introduced in the previous chapters, must be concerned with the development of applications for improving the human interaction with semantic models and data. Extensive research has been done on models, theoretical approaches and development of tools to infer new information from data and ontologies. However, the Semantic Web will only be fully integrated with the everyday-Web only when it will be easily available and fully accessible to every Web-users and not only to Semantic Web practitioners. This issue is even more crucial for Semantic Publishing and, in particular, for the legal domain since their end-users, publishers, researchers, librarians readers and legal experts are generally not competent on semantic technologies. These communities need to invest time and efforts in the development of proper user-friendly interfaces that act as intermediate between semantic models and end-users.

A good amount of work has been done in the past in this direction. For instance, ontology development editors were implemented (e.g., Protégé[1] (Knublauch et al. 2005) and the NeOn Toolkit (Suárez-ueroa et al. 2012)), Web search engines to look for semantic resources were launched (e.g., Sindice[2] (Oren et al. 2008) and Watson[3] (d'Aquin and Motta 2011)), and semantic desktop applications were released (e.g., SemNotes[4] (Dragan et al. 2011)). However, what the Semantic Publishing

[1] Protégé: http://protege.stanford.edu.

[2] Sindice: http://sindice.com.

[3] Watson: http://watson.kmi.open.ac.uk.

[4] SemNotes: http://smile.deri.ie/projects/semn.

S. Peroni, *Semantic Web Technologies and Legal Scholarly Publishing,*
Law, Governance and Technology Series 15, DOI 10.1007/978-3-319-04777-5_6,

community urgently needs specific tools to assist non-experts in semantic technologies in dealing with and publishing semantic data.

Generally, the process entails the following steps:

1. once ontologies suitable for a particular domain of interest have been found, the user needs (or wants) to *understand* these models with the minimum amount of effort;
2. then, the user will *develop* new models when existing vocabularies/ontologies are not able to fully describe the domain in consideration. The development process will require the interaction with domain experts and end-users so that a model that addresses the domain in consideration as close as possible can be produced;
3. finally, once agreed on the model to use, the user will *add* data accordingly and, furthermore, they might need to *modify* those data in the future.

Each of these four operations—understanding, developing, adding and modifying—should be supported by proper interfaces that simplify the work of users who are not expert in ontology-related formalisms and Semantic Web technologies.

In this chapter, I will describe influential work and my personal contribution in this area. I will introduce four different tools I developed—*LODE*, *KC-Viz*, *Graffoo* and *Gaffe*—so as to help users in dealing with semantic data and models, such as those described in the previous chapters. In addition, I will also discuss possible next steps to extend existing legislative editor by means of Semantic Web technologies and applications presented herein, and highlight the key characteristics of semantically-enhanced legislative editors of the future.

## 6.1 How to Help Users: Tools and Applications for Semantic Data

Semantic publishing end-users must be supported when choosing which ontology to adopt according to their needs. Obviously, *understanding* which ontology fits better within a particular domain is not so straightforward, in particular when users are not experts in ontology formalisms; the ontology has not a human-comprehensible *documentation*, or it is so large that it becomes difficult to quickly *make sense* of it. An additional complication is introduced when we need/want to develop new ontologies or to add/modify semantic data according to specific models. To this end, it is crucial to have functional interfaces that support the *creation* of an ontology and the *addition* of semantic data according to it.

In this section, I will introduce a series of works that try to address all the issues I listed above, namely: ontology documentation; ontology sense-making; visual modelling, and authoring tools for ontologies.

### 6.1.1 Ontology Documentation

The production of the natural language documentation of ontologies is an important and crucial part of any ontology development process. Such a documentation enable users to comprehend the extent of an ontology without being concern about the particular formal language used to define its axioms. At the same time, writing the documentation is an activity that requires an important amount of effort. Thus, in order to help authors of ontologies to document them, applications have been developed which create a first draft of the documentation starting from labels (i.e., *rdfs:label*); comments (i.e., *rdfs:comment*); different kinds of annotations (e.g., *dc:description*, *dc:creator*, *dc:date*), and the logical structure of the ontology itself.

*SpecGen*[5] is a Python tool for the generation of ontology specifications, released under the MIT license[6]. It is available as a standalone application and it has been used to prepare the HTML documentation of well-known ontologies, such as SIOC[7] (Bojars and Breslin 2010). SpecGen generates the documentation by processing an HTML template and adding the list of ontological classes and properties in specifiable positions which can be specified within that template. As result, we obtain a new HTML document where the natural language description of the ontology come entirely from the template made by authors, while the software takes care of adding all the information related to the logical structure of the ontology.

Contrarily to SpenGen that needs a base HTML template to work, *VocDoc*[8] is a (rather short) Ruby script that allows one to produce documentation starting from RDFS vocabularies and OWL ontologies and it is able to produce both HTML documents and LaTeX files containing the description of the ontology/vocabulary.

Like VocDoc, *OWLDoc*[9] is a fully-automatic generator of a set of HTML pages describing the target ontology. It organises the documentation of each ontological entity in different parts: the taxonomy involving the entity; the usage of this entity in the context of the ontology, and all the formal logical axioms related to the entity (in Manchester Syntax (Horridge and Patel-Schneider 2012)). OWLDoc has been developed as plugin of *Protégé*[10] (Knublauch et al. 2005) and as a Web application[11].

Oriented to Linked Data applications rather than to ontology documentation, *Paget*[12] is a PHP framework that, by getting an input URL through a browser, is able to dispatch the request according to the particular mime-type specified by the client. Paget returns RDF entities in four different formats: RDF; HTML; Turtle, and JSON.

---

[5] SpecGen: http://forge.morfeo-project.org/wiki_en/index.php/SpecGen.

[6] MIT license: http://www.opensource.org/licenses/mit-license.php.

[7] The Semantically-Interlinked Online Communities (SIOC) project: http://sioc-project.org.

[8] VocDoc: http://kantenwerk.org/vocdoc/.

[9] OWLDoc: http://code.google.com/p/co-ode-owl-plugins/wiki/OWLDoc.

[10] Protégé: http://protege.stanford.edu/.

[11] Ontology browser: http://code.google.com/p/ontology-browser/.

[12] Paget: http://code.google.com/p/paget.

It can be used to describe a set of pure RDF statements (*subject-predicate-object*)[13] and, to some extents, to produce an HTML human-comprehensible description from the axioms of an OWL ontology[14].

*Neologism*[15] (Basca et al. 2008) is a Web-based editor for the creation of RDFS vocabularies and (very simple) OWL ontologies. Moreover, it implements a publishing system that allows the publication of vocabularies and ontologies on the Web, rendered into natural language HTML pages. Basca et al.'s main objective is to reduce the time needed to create, publish and modify vocabularies for the Semantic Web.

Finally, *Parrot*[16] (Tejo-Alonso et al. 2011) is a Web service for the generation of HTML + Javascript documentation of OWL ontologies and RIF rules (Boley et al. 2013). This service allows one to specify multiple URLs identifying ontologies in order to produce an HTML summary "on the fly", starting from their logical structure and annotations.

### *6.1.2 Ontology Sense-Making*

The issue of how best to support visualisation and navigation of ontologies has attracted much attention in the research community. As Wang and Parsia (2006) emphasise, "effective presentation of the hierarchies can be a big win for the users", in particular, but not exclusively, during the early stages of a sense-making process, when a user is trying to build an initial mental model of an ontology and focuses less on the specific representational details than on the understanding of the overall organisation of the ontology. In particular, as discussed in Shneiderman (1996), there are a number of functionalities that an effective visualisation system needs to support, including (but not limited to) the ability to provide high level overviews of the data, to zoom in on specific parts of the data efficiently, and to filter out irrelevant details and/or irrelevant parts of the data.

One approach which allows high level overviews of hierarchical structures focuses on maximising the amount of information on display, through space-filling solutions, such as those provided by *treemaps* (Shneiderman 1992). Treemaps have proved to be a very successful and influential visualisation method, used not just to represent conceptual hierarchies but also to visualise information in several mainstream sectors, including news, politics, stock market, sport, etc. However, while treemaps define a clever way to provide concise overviews of very large hierarchical spaces, they are mostly effective when the focus is on leaf nodes and on a particular dimension of visualisation, in particular if colour-coding can be used to express different values for the dimension in question.

---

[13] Ian Davis' Linked Data profile, rendered through Paget: http://iandavis.com/id/me.html.

[14] A vocabulary for describing whisky varieties, rendered through Paget:http://vocab.org/whisky/terms.html.

[15] Neologism: http://neologism.deri.ie.

[16] Parrot: http://ontorule-project.eu/parrot/parrot.

However, as pointed out in Wang and Parsia (2006), treemaps are not very effective in supporting an understanding of topological structures, which is what is primarily needed in the ontology sense-making context. State of the art ontology engineering toolkits, such as *Protégé* (Knublauch et al. 2005) and *TopBraid Composer*[17], include visualisation systems that use the familiar node-link diagram paradigm to represent entities in an ontology and their taxonomic or domain relationships. In particular, both the *OwlViz visualizer* in Protégé and the "*Graph View*" in TopBraid, make it possible for users to navigate the ontology hierarchy by selecting, expanding and hiding nodes. However, OwlViz arguably provides more flexibility, by allowing the user to customise the expansion radius and supporting different modalities of use, including the option of automatically visualizing in OwlViz the current selection shown in the Protégé Class Browser.

*SpaceTree* (Plaisant et al. 2002), which also follows the node-link diagram paradigm, is able to maximise the number of nodes on display, by assessing how much empty space is available. At the same time, it also avoids clutter by utilising informative preview icons. These include miniatures of a branch, which are able to give the user an idea of the size and shape of an un-expanded subtree at a very high level of abstraction, while minimising the use of the available area.

Like treemaps, *CropCircles* (Wang and Parsia 2006) also uses geometric containment as an alternative to classic node-link displays. However, it tries to address the key weakness of treemaps, by sacrificing space in order to make it easier for users to understand the topological relations in an ontology, including both parent-child and sibling relations. The empirical evaluation comparing the performance of users on topological tasks using treemaps, CropCircles and SpaceTree introduced in Wang and Parsia (2006) showed that, at least for some tasks, users of CropCircles performed significantly better than those using treemaps. However, SpaceTree appears to perform significantly better than either treemaps or CropCircles on node finding tasks.

A number of hybrid solutions also exist, such as Jambalaya (Storey et al. 2001) and Knoocks (Kriglstein and Motschnig-Pitrik 2008), which attempt to combine the different strengths of containment-based and node-link approaches in an integrated framework, by providing both alternative visualisations and hybrid, integrated views of the two paradigms.

The group of techniques categorised in Katifori et al. (2007) as "context + focus and distortion" are based on "the notion of distorting the view of the presented graph in order to combine context and focus. The node on focus is usually the central one and the rest of the nodes are presented around it, reduced in size until they reach a point that they are no longer visible" (Katifori et al. 2007). These techniques are normally based on hyperbolic views of the data and offer a good trade-off—a part of the ontology is shown in detailed view, while the rest is depicted in the background. A good exemplar of this class of approaches is HyperTree (Souza et al. 2003).

Finally, I should also consider the most ubiquitous and least visual class of tools, exemplified by plugins such as the *Class Browser* in Protégé and the *Ontology*

[17] TopBraid Composer: http://www.topbraidcomposer.com.

*Navigator* in the *NeOn Toolkit*[18] (Suárez-ueroa et al. 2012). These follow the classic file system navigation metaphor, where clicking on a folder opens up its sub-folders. This approach is common in both file system interfaces and ontology engineering tools and, in the case of ontologies, it allows the user to navigate the ontology hierarchy simply by clicking on the identifier of a class, to display its subclasses, and so on. While superficially a rather basic solution, especially when compared to some of the sophisticated visual metaphors that can be found in the literature, this approach can be surprisingly effective for two reasons:

- it is very familiar to users;
- it makes it possible to display quite a lot of information in a rather small amount of space, in contrast with node-link displays, which can be space-hungry.

As a result it is not surprising that these interfaces often perform better in evaluation scenarios than the graphical alternatives. For instance, the evaluation reported in Katifori et al. (2006) shows that subjects using the Protégé Class Browser fared better than those using alternative visualisation plugins in a number of ontology engineering tasks.

### *6.1.3 Visual Modelling of Ontologies*

Usually, people who want to model ontologies have to be experts in formal languages (e.g., description logic) that are expressive enough to enable the definition of all the semantic constraints required by the particular needs of conceptualisation of a domain. As consequence of that, few years ago a new professional role arose to deal specifically with the development of ontologies: the *ontology engineer*.

Nowadays an increasing number of people in several fields (e.g., biology, medicine, literature, software engineering) are beginning to develop ontologies for their own needs. To this end, they usually adopt visual approaches that enable the development of complex ontologies while hiding the intrinsic complexity of the formal language they use. Moreover, as side effect, these visual approaches help when *presenting* ontologies to an audience that is not expert in formal languages.

One of the most common of these approaches is the *semantic networks* (Woods 1975). A semantic network is a "graphic notation for representing knowledge in patterns of interconnected nodes and arcs" (Sowa 1987). Ontology classes and individuals are defined as nodes of a graph (i.e., the visual representation of a semantic network). At the same time, direct and labelled arcs can interlink nodes to represent predicates between them: sub-class relations; belongingness to a class; individual assertions, etc. Although this tool is commonly used for representing OWL ontologies, it had been developed within the Artificial Intelligence field, years before the development of the Semantic Web.

[18] NeOn Toolkit: http://www.neon-toolkit.org.

Another interesting approach coming from software engineering fields is that of using modified versions of UML (Object Management Group 2011), opportunely adapted to describe OWL ontologies. Gasevic et al. (2004) propose a new UML profile for the definition of OWL ontologies that is entirely based on the UML class notation. Moreover, they illustrate a process that allows one to produce OWL ontologies by applying an XSLT transformation to the XMI version of an UML document made through an UML editor (such as Poseidon[19]).

Brockmans et al. (2004, 2006) propose another UML profile that enables one to define OWL entities using an extended set of UML-based graphic notations. In particular, the UML class notation is used for the representation of classes and, when opportunely appointed with dedicated stereotypes, class restrictions. Properties are represented as UML n-ary associations, and they might have stereotypes specified according to property characteristics (e.g., functional, symmetric, reflexive). Individuals belonging to particular OWL classes are represented by using the UML "object:Class" notation, while OWL data types are represented as UML classes.

In 2009, the industry consortium responsible of UML, i.e., the Object Management Group, released an official UML profile (Object Management Group 2009) for defining OWL ontologies, called *Ontology Definition Metamodel* (*ODM*), which incorporates and harmonises works done previously on this topic. All these UML-like proposals appear to be less intuitive than the semantic network approach, since they ask users to learn at least some basic principles of the UML notations. Of course, the UML-like approach has been very successful in software engineering communities, because it introduced, with a relatively low cost, software engineers to ontology modelling.

### *6.1.4 Authoring Tools for Ontologies*

Decorating resources (e.g., documents) with semantic data is usually a tedious task unless one has the support of appropriate applications that make this activity practicable, intuitive and relatively quick. Nowadays there exist many tools that help users to deal with metadata enrichment, such as metadata editors and automatic content processing mechanisms.

For instance, *DC-dot*[20] retrieves Web pages and automatically proposes related metadata according to Dublin Core Metadata Elements (Dublin Core Metadata Initiative 2012b) and Terms (Dublin Core Metadata Initiative 2012a). Metadata can be edited using the form provided by the system, which is also accompanied by a context-sensitive help. However, DC-dot only provides text areas for Dublin Core resources, and does not allow domain-specific constraints and any other kinds of customisations to the Dublin Core standard.

[19] Poseidon for UML: http://www.gentleware.com/products.html.

[20] DC-dot: http://www.ukoln.ac.uk/metadata/dcdot.

Another tool specifically designed for Dublin Core data is *Metamaker*[21]. This editor allows the creation of metadata from scratch through simple web forms. Metadata can be saved in different formats, e.g., HTML, XHTML, XML, RDF or AGRIS AP[22]. Unlike DC-dot, Metamarker allows one to use terms from external sources, such as the AGROVOC thesaurus[23].

*TKME*[24] is an application that allows the creation and modification of metadata, and organises them according to hierarchical structures (i.e., trees). Since TKME does not require the use of a particular metadata schema, it allows one to customise its interface according to alternative semantic models.

In contrast to TKME, *Metasaur* (Kay and Lum 2003) provides a general visualisation tool for ontologies describing a particular domain. Through Metasaur, users can create lightweight ontologies according to existing metadata schemas as well as change existing ontologies by adding new constraints and restrictions. As a drawback, however, Metasaur has not a flexible user interface. In fact, although it creates automatically a form starting from the ontology in consideration, it does not allow users to customise and personalise it in any way.

Some other approaches address metadata editing through Web applications, such as Wikis. Several approaches to semantic wikis have been developed to bring together the benefits of the free editing philosophy of wikis and ontological data. Semantic wikis can be organised into two main categories according to their connections with the ontologies: "wikis for ontologies" and "ontologies for wikis" (Buffa et al. 2008). In the first case, the wiki is used as a serialisation of the ontology: each concept is mapped into a page and typed links are used to represent object properties.

Such a model—initially proposed in the ontology for MediaWiki articles, called WikiOnt (Harth et al. 2005)—has been adopted by most semantic wikis. SemanticMediaWiki (Všlkel et al. 2006) is undoubtedly the most relevant one. It provides users with an intuitive syntax to embed semantics, i.e., RDF statements, within the markup of a page. *SemanticMediaWiki* allows users to edit the content freely without any limitation. The more the information is correctly encoded the more semantic data are available, but no constraint is imposed over the authoring process. SemanticMediaWiki has been developed with the idea of creating a machine-readable version of Wikipedia, to make full use of the huge amount of information and the competencies and enthusiasm of its community.

The original term *Wikitology* (Klein et al. 2005) summarises very well the potentialities of such approach. The DBPedia project (Auer et al. 2007) is also worth mentioning, being the most recent effort in translating the Wikipedia content into RDF. One of the main obstacles to the realisation of Wikitology and similar projects is without doubt the difficulty in creating semantic content. Although the syntax is very simple, in fact, authors still have to learn some new markup and above all they must

[21] Metamarker: http://www.fao.org/aims/tools/metamaker.jsp.

[22] AGRIS Application Profiles: http://aims.fao.org/standards/agmes/application-profiles/agris.

[23] AGROVOS thesaurus: http://aims.fao.org/website/AGROVOC-Thesaurus/sub.

[24] TKME: http://geology.usgs.gov/tools/metadata/tools/doc/tkme.html.

manually write each statement and the statements must be correct. *SemanticForms*[25] is an extension of SemanticMediaWiki that addresses such issue by allowing users to create semantic content via pre-defined forms. SemanticForms generates forms from templates, whose fragments and data have been previously typed. This process employs an embedded mapping between each datatype and each type of field (radio-buttons, checkboxes, textareas, etc.). Users do not need to write the statements manually anymore as they are only required to fill HTML forms.

The difficulties in generating SemanticMediaWiki data have been mitigated by an ad-hoc importer that allows the creation of multiple articles from an input OWL ontology (Vrandecic and Krötzsch 2006). This tool uses a PHP API for managing OWL and automatically creating wiki content according to the basic MediaWiki model: each concept is mapped into a page and each property into a typed link. The tool also carries out tests on the consistency of the ontology and any duplication of non-relevant data are also performed.

Other wikis provide users with mixed interfaces for creating semantic data. *MaknaWiki* (Dello et al. 2006), for example, is a JSP wiki-clone that allows users to embed semantic statements or to fill HTML forms for querying and adding data to the ontology represented by the wiki. These forms provide general tools for aided navigation of the semantic data, do not depend on the domain of the wiki and their structure is hard-coded in the system.

*Rhizome* (Souzis 2005) provides friendly interfaces and textareas where users can write statements directly. It relies on ZML (a textual syntax serialisable into XML), a generic language to express semi-structured data, and an engine to apply rules for intermixing semantics and free texts.

An original solution is provided by *AceWiki* (Bao et al. 2009) and *CNL-approach* (De Coi et al. 2009). AceWiki is a semantic wiki that allows users to write ontological statements by simply writing sentences in the ACE (Attempo Controlled English) language. The system includes a predictive authoring tool that suggests options to the users and autocompletes fields consistently to the ontology represented by the wiki. The same editor can be used to extend the ontology by creating new classes, instances and relations. In De Coi et al. (2009) De Coi *et al.* proposed a similar approach for SemanticMediaWiki, through a CNL (Controlled Natural Language) interface able to convert sentences written in multiple languages into semantic data.

The second category of semantic wikis, based on the principle of "ontologies for wikis", includes all those wikis that are built on top of ontological foundations. The idea is to exploit ontologies to create and maintain consistent semantic data within a wiki so that sophisticated analysis, queries and classifications can be performed on its content. *IkeWiki* (Schaffert 2006) was one of the first wikis to adopt this approach. This tool works by loading an OWL ontology into the system which is then automatically translated into a set of wiki pages and typed links. Multiple interfaces are provided to the users for editing the plain wiki content, adding new metadata or tagging pages. IkeWiki strongly relies on Semantic Web technologies: it

[25] SemanticForms: http://www.mediawiki.org/wiki/Extension:Semantic_Forms.

even includes a Jena OWL[26] (Carroll et al. 2004) repository and a SPARQL (Garlik and Seaborne 2013) engine used for navigation, queries and display of the semantic content of the wiki.

Similarly, OntOWiki (Auer et al. 2006) is a complete ontology editor. It relies on a strong distinction between the ontological back-end of the system and a user-friendly interface. Data are stored as OWL/RDF statements that are dynamically rendered into the final HTML wiki pages. OntOWiki provides users with multiple views of the same content: (1) ontological data can be navigated by listing classes, individuals, properties, etc.; (2) domain-specific views can be added as plugins (for instance, a MapView of geographical data can be dynamically mashed-up form GoogleMaps), and (3) editing-views are available in the system.

*SweetWiki* (Buffa et al. 2008) implements a user-friendly ontology tool designed for both expert and non-expert users. Two aspects characterise the system: the strong connection with the ontologies and the provision of Ajax-based interfaces for editing content and metadata. SweetWiki defines a "Wiki Object Model", i.e., an ontology describing the wiki structure. Concepts like "document", "page", "link", "version", "attachment" are all codified in an OWL file that is accessed and manipulated through the wiki itself. These concepts are made explicit in SweetWiki, although they are usually hard-coded in most semantic wikis. SweetWiki also allows users to import external ontologies and to access and manipulate those ontologies through ad-hoc interfaces (similar to those provided by the above mentioned full ontology editors). Finally, the system provides "assisted social tagging" facilities: users can add metadata to any page and can put pages in relation. These metadata values form a folksonomy that, on the one hand, is freely editable by users and, on the other, is built on top of ontological data. The interface for tagging, in fact, suggests consistent metadata by exploiting SPARQL queries and autocompletion features.

Finally, UFOWiki (Passant and Laublet 2008) is another project that aims at integrating wikis, ontologies and forms. UFOWiki is a wiki farm, i.e., a server that allows users to set up and deploy semantic wikis. The overall content is stored in a centralised repository as RDF triples that express both the actual content of each page and its metadata. The same farm deploys multiple wikis, so that they can share (ontological) data in a distributed environment.

## 6.2 LODE: Generating HTML Documentation from Ontologies

Usually, when someone wants to understand the extent of a particular ontology, the first thing to do is to look for the ontology's human-readable documentation. A large number of ontologies, especially those used in the Linked Data, have a very good and comprehensive Web page describing theoretical backgrounds and developed entities.

Problems arise in relation to underdeveloped models, since natural language documentation is usually published only when an ontology becomes stable. This

[26] Jena: http://jena.sourceforge.net.

approach is justifiable: writing a proper documentation requires a big effort and changing it every time the ontology is modified can be problematic.

An additional difficulty is given by the availability of several "stable" ontologies that do not have any document describing them. Thus, the only way to understand them is to use an ontology editor to explore their logical axioms. This approach may however create problems to people approaching the ontology world for the very first time. Firstly, they would have to download and install the ontology editor, if not already present on their machine. Secondly, they would have to learn how to use the editor. And finally, they will be able to get a sense of the ontology by loading it onto the editor.

Obviously, this process can be very time-consuming. In order to address this issue, standalone tools and Web applications have been developed, as illustrated in Sect. 6.1.1. However they lack a proper and quick mechanism for the conversion of all the ontology axioms into a human-readable documentation.

To address this issue, I developed the *Live OWL Documentation Environment*[27] (*LODE*) (Peroni et al. 2012). It is a service that automatically extracts classes, object properties, data properties, named individuals, annotation properties, meta-modelling (punning), general axioms, SWRL rules and namespace declarations from any well-formed OWL or OWL 2 ontology, and returns them as ordered lists, together with their textual definitions, in a human-readable HTML page designed for browsing and navigation by means of embedded links.

LODE is basically based on an XSLT stylesheet that takes RDF/XML linearisation of an ontology (Beckett 2004), produced through the OWLAPI[28] (Horridge and Bechhofer 2011), as input and converts it into an HTML representation. If the target ontology is already linearised in that format, it is then possible to call the service specifying directly its URL (i.e., "http://www.essepuntato.it/lode/") followed by the complete URL of the ontology. For instance:

```
http://www.essepuntato.it/lode/http://www.essepuntato.it/2008/12/earmark
```

In the following subsections I will introduce the most important features of LODE.

### 6.2.1 What Axioms are Used to Create the Documentation

Primarily, LODE interprets the most common annotation properties used for the description of entities, in particular[29]: *dc:contributor*; *dc:creator*; *dc:date*; *dc:description*; *dc:rights*; *dc:title*; *dcterms:contributor*; *dcterms:creator*; *dcterms:date*; *dcterms:description*; *dcterms:rights*; *dcterms:title*; *owl:versionInfo*;

[27] LODE, the Live OWL Documentation Environment: http://lode.sourceforge.net.

[28] OWLAPI: http://owlapi.sourceforge.net.

[29] The prefixes *dc*, *dcterms*, *owl* and *rdfs* in the following list respectively refers to "http://purl.org/dc/elements/1.1/", "http://purl.org/dc/terms/", "http://www.w3.org/2002/07/owl#" and "http://www.w3.org/2000/01/rdf-schema#".

*rdfs:comment*; *rdfs:isDefinedBy*, and *rdfs:label*. LODE adopts the following rules when transforming those annotations into HTML documentation:

- in the presence of Dublin Core annotations defined according to both DC Metadata Elements (Dublin Core Metadata Initiative 2012b) and DC Metadata Terms (Dublin Core Metadata Initiative 2012a), the former have precedence;
- dates (i.e., *dc:date* and *dcterms:date*) written according to the XML Schema datatype (i.e., *yyyy-mm-dd*) are automatically transformed into *dd/mm/yyyy*;
- agents (i.e., *dc:creator*, *dc:contributor*, *dcterms:creator* and *dcterms:contributor*) are rendered either as strings or as clickable URLs according to their types, i.e., literals or resources, respectively;
- descriptions (i.e., *dc:description* and *dcterms:description*) are rendered either as strings or as media objects according to their types, i.e., literals or resources, respectively;
- comments (i.e., *rdfs:comment*) and descriptions (i.e., *dc:description* and *dcterms:description*) are represented, respectively, as abstracts and as detailed descriptions of entities;
- labels (i.e., *rdfs:label*) and QNames (when labels are not specified) are used to refer to all the entities of the ontology, instead of using their URLs;
- the nature of each entity is identified by a descriptive abbreviation, according to its type: "c", "op", "dp", "ap" and "ni" are used to identify class, object property, data property, annotation property and named individual, respectively.

Figures 6.1 and 6.2 on page 207–208 show how these annotations are rendered for an example ontology, EARMARK (Di Iorio et al. 2011b), an ontology that defines entities which describe document markup (such as elements, attributes, comments and text nodes).

LODE converts all the other axioms of the ontology into Manchester Syntax definitions (Horridge and Patel-Schneider 2012), as shown in Fig. 6.3 on page 208. I prefer to use this syntax rather than any other because it is the most human-comprehensible syntax for ontological axioms, and thus the most helpful for non-specialists.

Ontological axioms are rendered in grey boxes, one for each entity declared in the ontology. The axioms taken into account by LODE refer to: super-class and super-property; equivalent class and property; disjoint class and property; property domain and range; property chain; keys; object/data property assertion; type; imported ontology; generic axiom, and SWRL rule. Moreover, LODE automatically enriches those definitions, by adding information about sub-classes, domain/range properties of classes, sub-properties and entity meta-modelling.

### *6.2.2 Special Parameters to Call the Service*

LODE can be invoked with a number of optional parameters in order to limit or extend the final documentation that is produced. For instance, it is possible to take

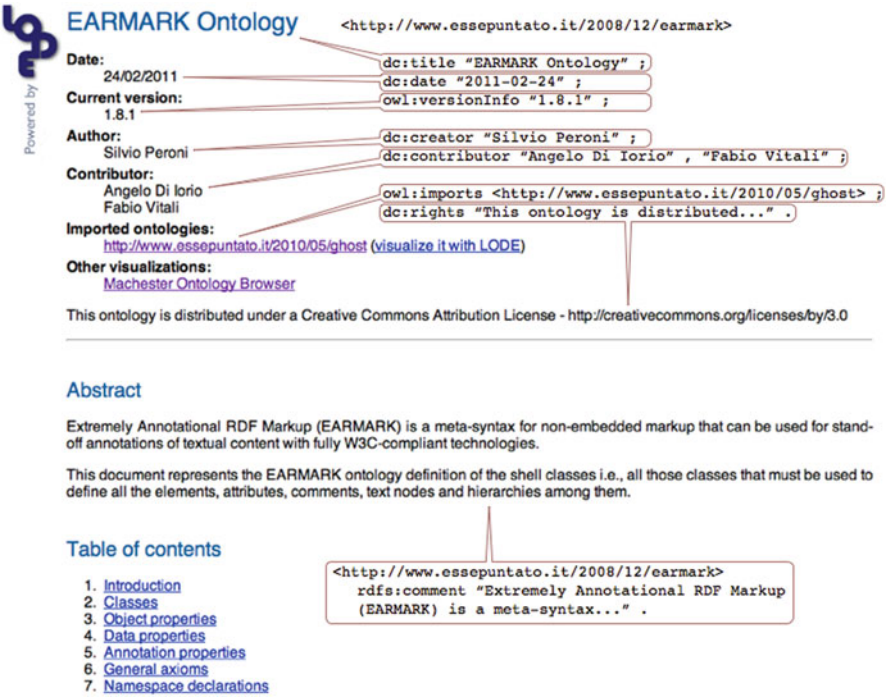


**Fig. 6.1** The beginning of the Web page generated by LODE for the EARMARK Ontology, annotated with OWL assertions in Turtle (not present in the normal LODE web page) illustrating how these assertions are rendered in HTML

into account all the entities in the ontology closure and/or the inferred axioms. The following pseudo-URL describes how to call LODE:

http://*www.essepuntato.it/lode/optional-parameters/ontology-url*

In particular:

- *www.essepuntato.it/lode* is the URL to call the service;
- *ontology-url* is the full "http:// . . . " URL of the OWL ontology that will be processed by the service. It must be always the last item of the pseudo-URL, and may be preceded by one or more (slash-separated) parameters.

Figure 6.4 on page 209 illustrates the alternative ways to build the URL to call LODE and the related modules used. The optional slash-separated parameters are described in the following sub-sections.

#### 6.2.2.1 Parameter "Owlapi"

When this optional parameter is specified, the ontology defined in *ontology-url* will be pre-processed via OWLAPI (Horridge and Bechhofer 2011), in order to linearise it in the RDF/XML format accepted by LODE. The use of this parameter is strongly

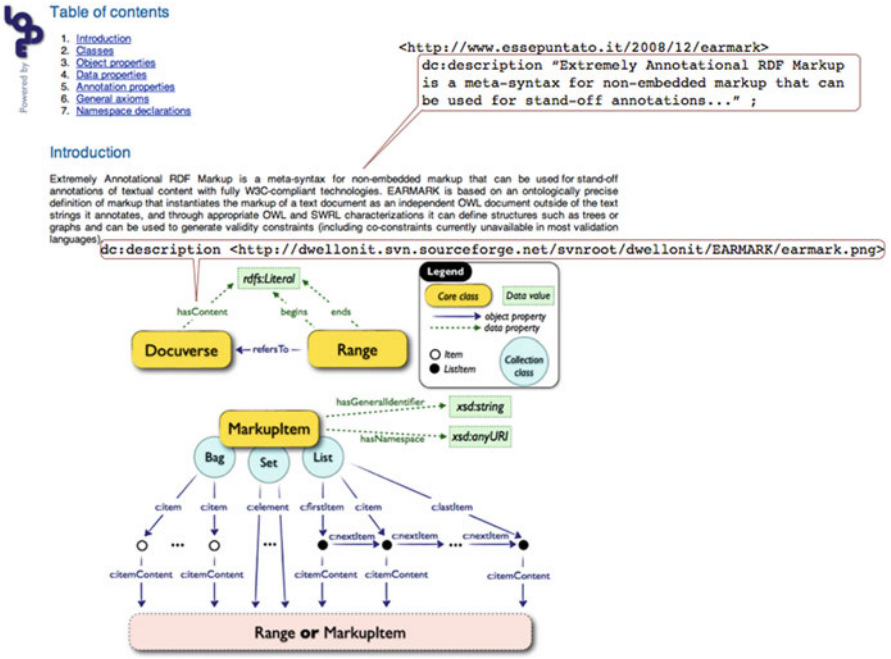


**Fig. 6.2** Two possible kinds of descriptions: pure string (for literals) and media object (for resources).

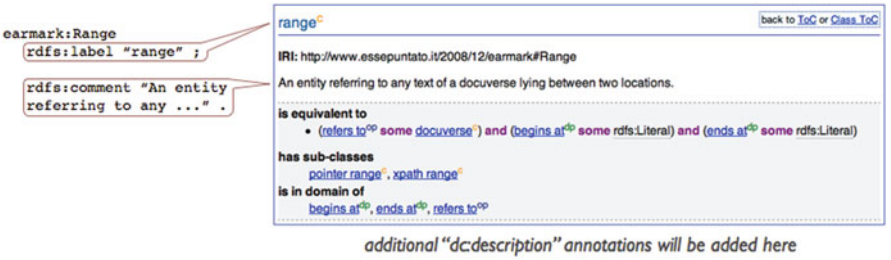


**Fig. 6.3** How entities (classes, properties and individuals) are rendered by LODE

recommended: it allows LODE to process ontologies implemented in all the formats supported by the OWLAPI.

#### 6.2.2.2 Parameter "Imported"

When this optional parameter is specified, the axioms in the imported ontologies of *ontology-url* are added to the HTML description of the ontology. This parameter implicitly specifies the *owlapi* parameter.

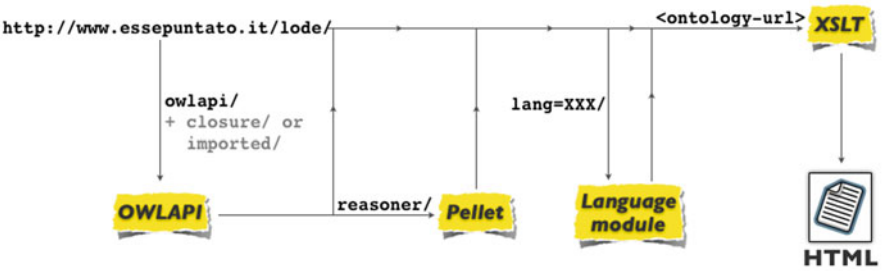


**Fig. 6.4** All the possible ways, according to specific needs, for making a request to LODE

#### 6.2.2.3 Parameter "Closure"

When this optional parameter is specified, the transitive closure given by considering the imported ontologies of *ontology-url* is added to the HTML description of the ontology. This parameter implicitly specifies the *owlapi* parameter. If both the parameters *closure* and *imported* are specified (in any order), *imported* will be preferred.

#### 6.2.2.4 Parameter "Reasoner"

When this optional parameter is specified, the inferred axioms of *ontology-url* (through the Pellet reasoner (Sirin et al. 2007)) will be added to the HTML description of the ontology. This parameter implicitly specifies the *owlapi* parameter. Note that, depending on the nature of the ontology to process, this computationally intensive function can be very time-consuming.

#### 6.2.2.5 Parameter "lang"

When this optional parameter is specified, the selected language will be used as preferred language instead of English when showing annotations of *ontology-url*. It must be followed by an "=" and the abbreviation of the language to use. e.g.,: "lang = en" for English, "lang = it" for Italian, "lang = fr" for French, etc.

### *6.2.3 URI Fragments*

LODE offers intuitive mechanisms to refer to particular ontological entities within the HTML documentation, according to the URL of the entity in consideration. The following extension of the pseudo-URL introduced in Sect. 6.2.2 defines how to refer to a particular entity of an ontology:

http://www.essepuntato.it/lode/optional-parameters/ontology-url *#entity*

For instance, to generate the documentation of FaBiO (Sect. 5.1) and then jumping directly to the point where the resource "http://purl.org/spar/fabio/Article" is described, I need to invoke LODE as follows:

```
http://www.essepuntato.it/lode/http://purl.org/spar/
   fabio#http://purl.org/spar/fabio/Article
```

This request can be simplified by looking for descriptions of entities defined as *fragment* of the ontology's URL, such as the entity *Element* of the EARMARK ontology—i.e., "http://www.essepuntato.it/2008/12/earmark#Element". In this particular case, I can use either the entire entity URL as illustrated previously or the entity's local name only, as shown here below:

```
http://www.essepuntato.it/lode/http://www.essepuntato.it
   /2008/12/earmark#Element
```

### 6.2.4 Content Negotiation via .htaccess

LODE can be freely used by third parties, as described in its documentation. In particular, it can be very useful if used in conjunction with content negotiation mechanisms to display a human-readable version of an OWL ontology when the user accesses the ontology using a web browser, or to deliver the OWL ontology file itself when the user accesses the ontology using an ontology development tool such as *Protégé* (Knublauch et al. 2005) or the *NeOn Toolkit* (Suárez-ueroa et al. 2012). For instance, an implementation of such a content negotiation is given in Berrueta and Phipps (2008) by using the *.htaccess* file:

```
AddType application/rdf+xml .rdf

# Rewrite engine setup
RewriteEngine On

# Rewrite rule to serve HTML content
RewriteCond %{HTTP_ACCEPT} !application/rdf\+xml.*(text/
   html|application/xhtml\+xml)
RewriteCond %{HTTP_ACCEPT} text/html [OR]
RewriteCond %{HTTP_ACCEPT} application/xhtml\+xml [OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/.*
RewriteRule ^ontology$ http://www.essepuntato.it/lode/
   http://www.mydomain.com/ontology [R=303,L]

# Rewrite rule to serve RDF/XML content if requested
RewriteCond %{HTTP_ACCEPT} application/rdf\+xml
RewriteRule ^ontology$ ontology.owl [R=303]

# Choose the default response
RewriteRule ^ontology$ ontology.owl [R=303]
```

LODE can be seen in action by opening, in a Web browser, any of ontologies presented in this book. For instance, the URL "http://purl.org/spar/fabio" resolves,

**Table 6.1** The five tasks subjects performed in the user testing session

| | |
|---|---|
| *Task 1* | Describe the main aim of the ontology |
| *Task 2* | Describe what the class doctoral thesis defines |
| *Task 3* | Describe what the object property has subject term describes, and record its domain and range classes |
| *Task 4* | Record the class having the largest number of direct individuals (i.e., individuals that belongs explicitly to that class and that are not inferable from its subclasses) |
| *Task 5* | Record all the subclasses and properties involving the class item |

by content negotiation, to display the LODE HTML version of the FaBiO ontology with the URL "http://www.essepuntato.it/lode/http://purl.org/spar/fabio". As shown previously, a similar syntax can be used to display the LODE visualisation of any other OWL ontology.

### *6.2.5 Test and Evaluation*

In order to gather data about the usability of LODE, I carried out user testing. I asked 13 subjects to perform five unsupervised tasks (max. 5 min/task), involving ontology navigation through LODE documentation. There were no "administrators" observing the subjects while they were undertaking these tasks. All the subjects were volunteers who responded to personal e-mails or to an invitation sent to the semantic-web@w3.org and public-lod@w3.org mailing lists.

For the tasks, I used a *medium-size* ontology, namely FaBiO, the *FRBR-aligned Bibliographic Ontology* (see Sect. 5.1), which, at the time of the test, was composed by 214 classes, 69 object properties, 45 data properties and 15 individuals. FaBiO was chosen because we expected most people involved in the experiments (primarily Semantic Web researchers and practitioners) to have familiarity with the domain it describes, i.e., bibliographic entities such as research papers, journal articles and books. In addition, FaBiO was also chosen because using an ontology larger than FaBiO would have required more time to complete the tasks, potentially reducing the number of users willing to complete the test, and thus reducing the number of useful data for the evaluation.

The tasks given to the subjects are shown in Table 6.1. This set of tasks was designed to exploring the LODE capabilities in creating a human-readable documentation and in browsing ontologies.

Task 1 is a pure descriptive activity that involves only the documentation produced by LODE, without using any navigational features such as Web links. Tasks 2 and 3 are similar to Task 1, but in addition, they typically require the user to use some navigational facilities to reach the class *doctoral thesis* and the object property *has subject term*. Finally, Tasks 4 and 5 further assess how easily LODE enables users to browse the ontology and understand its structure.

My aim was to assess how well LODE would help users, by producing human-readable documentation of an OWL ontology, which would enable them to browse and make sense of it easily.

**Table 6.2** System usability scale values and related sub-measures

| Measure | Mean | Max. value | Min. value | Standard deviation |
|---|---|---|---|---|
| SUS value | 77.7 | 92.5 | 57.5 | 12.5 |
| Usability | 76.4 | 90.6 | 56.3 | 12.8 |
| Learnability | 82.7 | 100 | 62.5 | 14.9 |

The test session was structured as follows. I first asked the subjects to complete a short multiple-choice questionnaire about their background knowledge and skills in OWL, ontology engineering, and ontology documentation (max. 2 min). Then, as a warm-up task, I asked the subjects to use LODE to explore the FOAF ontology[30], a relatively simple ontology, in order to become familiar with the structure of the documentation it produced, and with its navigation mechanisms (primarily, internal hypertext links) (max. 5 min). Then, for the test proper, I asked the subjects to complete the five tasks listed in Table 6.1, by using the documentation of the FaBiO ontology created by LODE (ideally 2, max. 5 min/task). Finally, I asked the subjects to fill in two short questionnaires, one multiple choice and the other textual, to report their experience of using LODE in completing these tasks (max. 5 min). All the questionnaires and all the outcomes of the experiments are available online[31].

Out of the 65 tasks in total (5 tasks given to each of 13 subjects), 58 were completed successfully (i.e., the right answer was given), while 7 had incorrect answers or were not completed at all, giving an overall success rate of 89 %. The 58 successes were distributed as follows: 13 (out of 13) in Task 1, 13 in Task 2, 13 in Task 3, 10 in Tasks 4 and 9 in Task 5.

The usability score for LODE was computed using the *System Usability Scale* (*SUS*) (Brooke 1996), a well-known methodology used for the perception of the usability of a system. It has the advantage of being technology independent (it has been tested on hardware, software, Web sites, etc.) and it is reliable even with a very small sample size (Sauro 2011). In addition to the main SUS scale, we were also interested in examining the sub-scales of pure *Usability* and pure *Learnability* of the system, as proposed by Lewis and Sauro (2009). As shown in Table 6.2, the mean SUS score for LODE was 77.69 (in a 0–100 range), abundantly surpassing the target score of 68 which indicates a good level of usability (Sauro 2011). The mean values for the SUS sub-scales Usability and Learnability were 76.4 and 82.7, respectively. In addition, two sub-scores were calculated for each of the subjects by considering the values of the answers they had given in the background questionnaire, which was composed of ten questions about the subject's experience with ontologies and two questions about their experience with ontology documentation tools. We then compared these sub-scores with the SUS values and the other sub-scales using the Pearson's $r$, and found weak negative correlation (between $-0.34$ and $-0.14$) between the experience sub-scores and the SUS values. This may show that the perceived usability of LODE does not depend upon any particular experience of

[30] FOAF ontology: http://xmlns.com/foaf/spec/index.rdf.

[31] http://www.essepuntato.it/2012/04/lodeusertesting.

**Table 6.3** Terms—three positive (+) and two negative (−)—mentioned by more than one individual in the final questionnaire responses

| Term | Description | Frequency |
|---|---|---|
| Search (−) | No search function was provided to directly look for and access entities of the ontology. Users acknowledge that since the ontology is on a single web page, they could use (and in fact did use) the search function of the browser, but many still found it a missing feature | 7 out of 11 |
| Readability (+) | High praise was given to the clarity of the presentation, the intuitiveness of the organisation, and the immediacy of identifying the sought information. The good typographical style of the output is clearly among the best qualities of LODE | 5 out of 11 |
| Links within the document (+) | The systematic use of internal links to the various features of the ontology was considered useful and immediately usable | 4 out of 11 |
| Scalability (−) | The LODE interface provides no links to entities provided by external, but linked ontologies. A highly modular ontology composed of a multiplicity of independent sub-ontologies is hard to navigate, and similarly the structure on a single page could make really large ontologies quite hard to access | 3 out of 11 |
| Single page (+) | Praises were given to the idea of placing all the content on a single Web page, which allows a multiplicity of approaches to accessing and reading the ontology, including visual transitions and scrolling that would not be possible if the ontologies had been presented in separate web pages | 2 out of 11 |

the subjects in the use of ontologies and ontology documentation tools, rather the opposite. However, each correlation measure appears to be not statistically significant and we need to enrich our dataset to come to a more definite conclusion.

Axial coding of the personal comments expressed in the final questionnaires (Strauss and Corbin 1998) revealed only a small number of commonly perceived issues. Out of 13 which participated to the test, 11 subjects had comments that were meaningful for the study, and, of the 15 terms that were identified as significant, only 5 (3 positive and 2 negative) were mentioned by more than one individual (albeit sometimes with different words), as shown in Table 6.3.

Additional comparative testing sessions, which involved LODE and other Web-available ontology documentation (i.e., *Parrot* and *Ontology Browser*) and ontology engineering tools (*WebProtégé*), confirm these results. As described in Peroni et al. (2013), these latter sessions showed that LODE was perceived more usable than other tools by users when carrying out tasks involving the understanding and browsing of OWL ontologies.

## 6.3 KC-Viz, a Tool for Visualising and Navigating Ontologies

Sometimes, the HTML documentation may not be enough to understand an ontology properly. This is particularly true for very large ontologies that make it difficult for a person to try to understand their overall structure. Having a clear natural language

documentation may not adequately support the users. In this case, using abstraction mechanisms that try to take a first meaningful snap-shot of the overall organisation of the ontology can be extremely useful. Starting from this summary, that may contain for instance the most representative concepts of the ontology, one can start to *make sense* of the ontology itself.

When I speak about "making sense" of an ontology, I refer to a specific ontology engineering task, where the user is primarily concerned with understanding the contents and overall structure of the ontology, i.e., acquiring an overview of the concepts covered by the ontology and the way they are organised in a taxonomy. Thus, the sense-making process includes:

- understanding the overall size[32] and shape[33] of the ontology;
- identifying the main components of the ontology and the typical exemplars of these components. Informative exemplars can also help the user to predict the siblings of the class (i.e., the exemplar) in question, thus playing a summarisation role not just with respect to its subtree, but also with respect to its siblings.

Of course, users need to be supported by interfaces and effective summarisation techniques when trying to make sense of large ontologies. That is because, once an ontology is large enough, it is not possible to show its entire structure in the limited space provided by a computer screen and therefore a difficult decision needs to be addressed. On the other hand the information on display needs to be coarse-grained enough to provide an overview of the ontology, thus ensuring the user can maintain an overall mental model of the ontology.

In this scenario, an exploration process needs to be supported, where the user can effectively home in on parts of the ontology, thus changing the level of analysis, while at the same time not losing track of the overall organisation of the ontology.

However, a problem affecting all the approaches discussed in Sect. 6.1.2 is that they all essentially use geometric techniques to providing abstraction. In contrast with these approaches, human experts are able to provide effective overviews of an ontology, simply by highlighting the key areas covered by the ontology and the classes that best describe these areas. In particular, the work reported in (Peroni et al. 2008), that I briefly introduce in Sect. 6.3.1, provides empirical evidence that there is a significant degree of agreement among experts in identifying the main concepts in an ontology. It also shows that the algorithm presented for *key concept* extraction (*KCE*) is able to retrieve a summarisation of the ontology maintaining the same level of agreement with the experts (Peroni et al. 2008).

---

[32] Given a node in the ontology, its *size* is the total number of its direct and indirect subclasses.

[33] Given a node in the ontology, its *shape* is an indication of the organisation of the subclasses. For instance, an ontology (or part of it) can have a horizontal (i.e., many subclasses and few levels of depth), or a vertical (i.e., many inheritance levels and only a few subclasses at each level) shape (Tartir et al. 2006). Understanding the shape of an ontology (or part of it) also means to understand whether it is balanced, indicating that all parts of the (sub-)ontology in question have been developed to a similar extent, or unbalanced, possibly indicating that some parts of the (sub-)ontology are less developed than others.

In this section, I will introduce KC-Viz, the *key concept visualiser* (Motta et al. 2010, 2011a, b, 2012), a tool for ontology visualisation and browsing I developed in collaboration with professor Enrico Motta and his research group (Open University, UK). Building on its ability to abstract out from large ontologies through the KCE algorithm, KC-Viz provides a rich set of navigation and visualisation mechanisms, including flexible zooming into and hiding of specific parts of an ontology, history browsing, saving and loading of customised ontology views, as well as essential interface customisation support, such as graphical zooming, font manipulation, tree layout customisation, and other functionalities. KC-Viz is a core plugin of the NeOn Toolkit (Suárez-ueroa et al. 2012).

In the following sections I will give a general view of the KCE algorithm principles and of the main features implemented in KC-Viz. Moreover, I will also report on additional findings gathered through questionnaires, which offer a number of other insights.

### *6.3.1 Key Concept Extraction*

Informally, *key concepts* can be seen as the best descriptors of an ontology, i.e., information-rich concepts, which are most effective in summarising what an ontology is about. In Peroni et al. (2008) a number of criteria are considered to identify the key concepts in an ontology, which are introduced as follows.

**Natural Category** KCE uses the notion of *natural category* (Rosch 1978)[34], to identify concepts that are information-rich in a psycho-linguistic sense. This notion is approximated by means of two operational measures: *name simplicity*, which favours concepts that are labelled with simple names, and *basic level*, which measures how "central" a concept is in the taxonomy of an ontology.

**Density** The notion of *density* highlights concepts that are information-rich in a formal knowledge representation sense, i.e., they have been richly characterised with properties and taxonomic relationships. The density is decomposed in two subcriteria, *global* and *local* density. While the global measures are normalised with respect to all the concepts in the ontology, the local ones consider the relative density of a concept with respect to its surrounding concepts. The aim here is to ensure that "locally significant" concepts get a high score, even though they may not rank too highly with respect to global measures.

[34] Eleanor Rosch has been one of the pioneers of the *prototype theory*, i.e., a cognitive approach for categorisation where some members of a particular category are considered more "central" than others. Rosch's works on categorisation have been the basis of significant subsequent research, such as the notion of *conceptual spaces* by Gardenfors (2000) and of *conceptual metaphors* by Lakoff (1987), Lakoff and Johnson (1980). In this work I have used an operative definition of Rosch's *natural category* derived from her notion of *basic level*.

**Coverage** The notion of *coverage* is used to ensure that no important part of the ontology is neglected, by maximising the coverage of the ontology with respect to its taxonomic relationships (*rdfs:subClassOf*).

**Popularity** The notion of *popularity*, drawn from lexical statistics, is introduced as a criterion to identify concepts that are likely to be most familiar to users. Similarly to the density criterion, the popularity is decomposed in two sub-criteria, global and local popularity.

Each of these seven criteria produces a score for each concept in the ontology and the final score assigned to a concept is a weighted sum of the scores resulting from individual criteria. As described in Peroni et al. (2008), which provides a detailed account of our algorithm and a formal definition of the criteria it employs, the approach has been shown to produce ontology summaries that correlate significantly with those produced by human experts.

It is important to emphasise that in the current online version of KCE[35] and in KC-Viz, the popularity criterion is not longer used, because of its computational cost on large ontologies. However, on the basis of the analytical studies described in Li et al. (2010), the weights associated with the other criteria has been parameterised, to produce a vastly more efficient version, while at the same time maintaining the same level of compliance with respect to the available human-generated benchmarks as the version of the algorithm presented in Peroni et al. (2008).

### *6.3.2 KC-Viz Main Features*

Running KC-Viz on an ontology (e.g., FaBiO) for the first time produce an initial visualisation of the network of classes, which includes concepts at different levels in the class hierarchy. The visualisation in Fig. 6.5 includes 26 concepts because I set the size of our ontology summary to 25 and the algorithm has automatically added the most generic concept, *owl:Thing*, to ensure that the visualisation displays a connected graph.

If we wish to display more or less succinct graphs, we can do so by changing the size of the ontology summary. The solid grey arrows in Fig. 6.5 indicate direct *rdfs:subClassOf* links, while the dotted green arrows indicate indirect *rdfs:subClassOf* links.

Nevertheless, starting from the first visualisation, KC-Viz provides the size of the tree under a particular class, which is indicated by a pair of integers, referring to the number of direct and indirect subclasses. For instance, Fig. 6.5 tells us that class *Endeavour* has 4 direct subclasses and 191 total subclasses (direct + indirect). Although more exploration is obviously needed to get a thorough understanding of the content of the FaBiO ontology, it can be argued that as a first step, the visualisation

[35] KCE Live: http://www.essepuntato.it/kce.

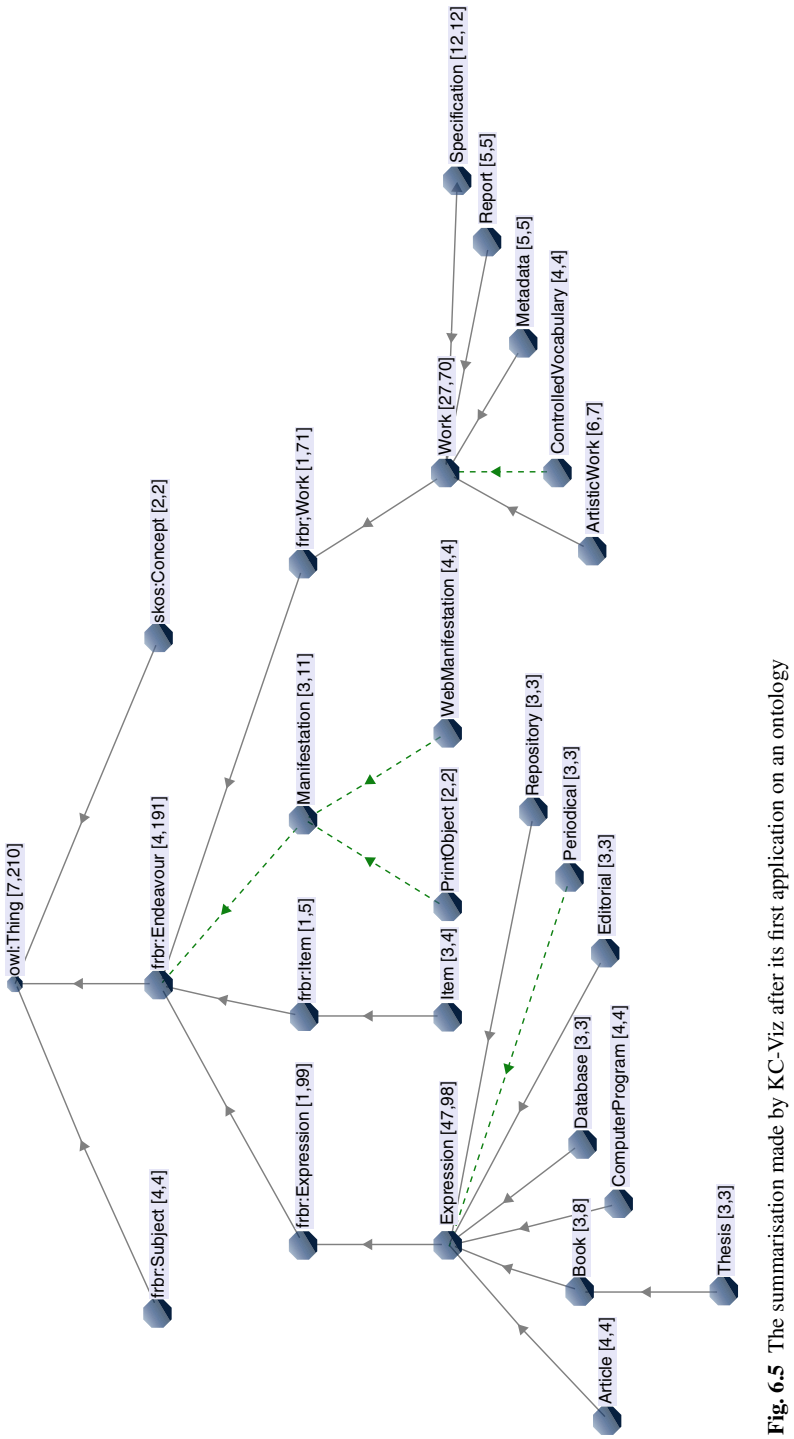


**Fig. 6.5** The summarisation made by KC-Viz after its first application on an ontology

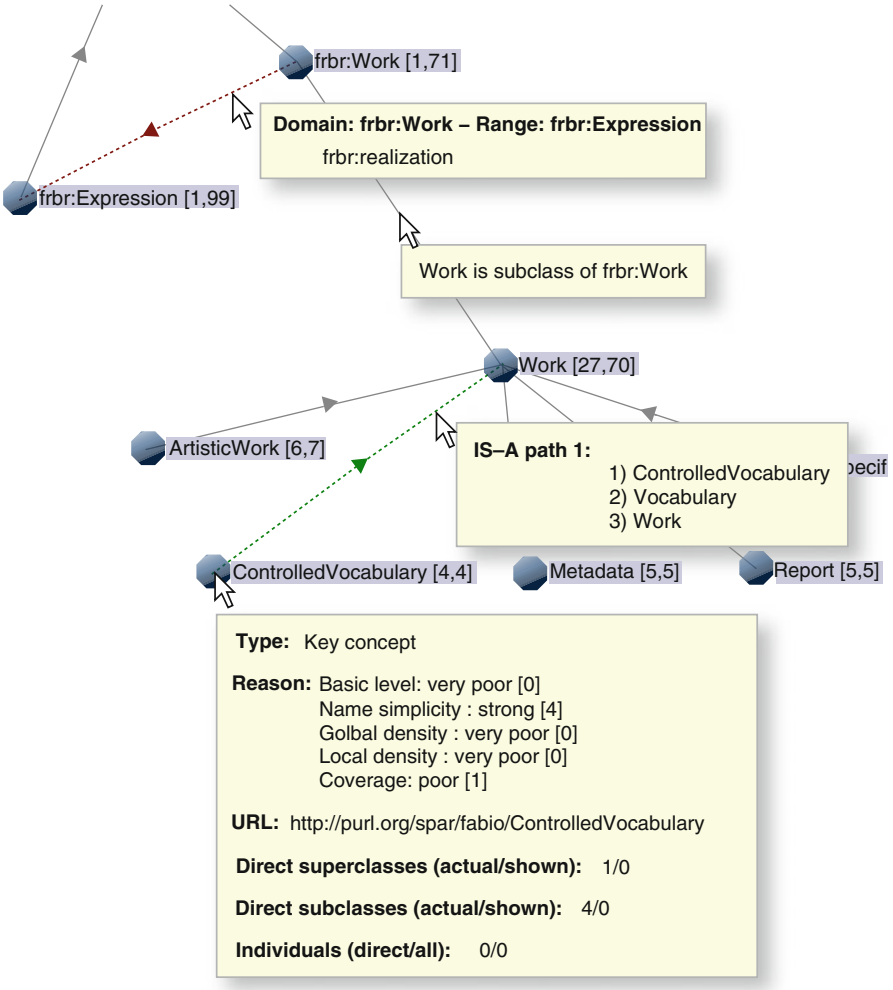


**Fig. 6.6** Tooltips that appear hovering nodes and edges

shown in Fig. 6.5 already provides a rather effective starting point for the ontology sense-making process.

#### 6.3.2.1 Description of Nodes and Arcs

By hovering the mouse over an element (node or edge) of the tree, as shown in Fig. 6.6 on page 218, a tooltip is popped up with some information about the element itself: the chain of *rdfs:subClassOf* relations or of domain/range links for edges, as well as information about number of (shown and total) sub- or super- classes for nodes. For

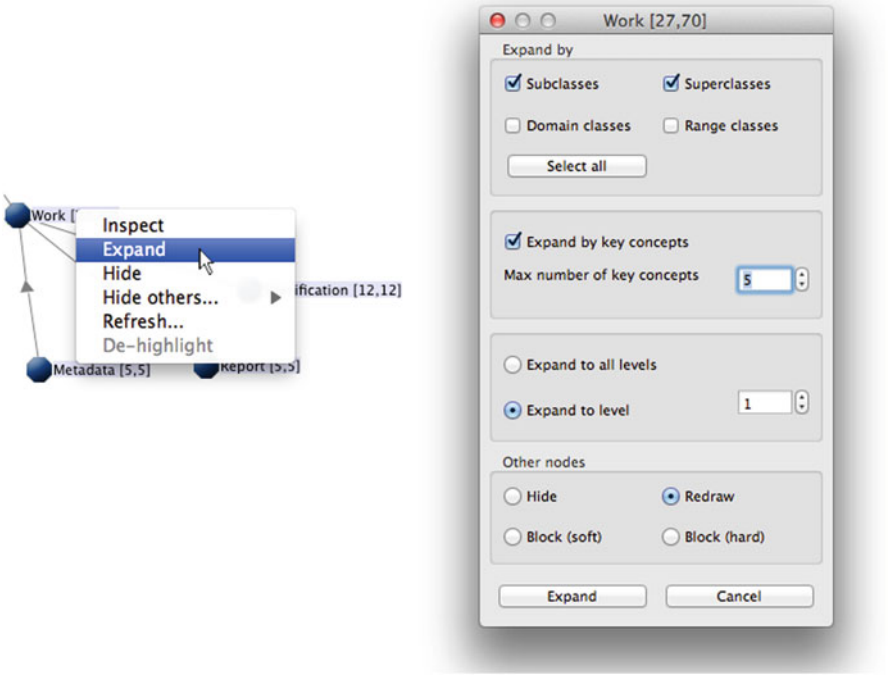


**Fig. 6.7** The menu popped up after clicking on the "Expand" option

the latter elements, there exists also a section, called "Reason", which indicates how a node (i.e., a class) fares with respect to the criteria used to determine key concepts. KC-Viz uses a 0–5 scale of labelled values (0 very poor, 1 poor, 2 fair, 3 good, 4 strong, 5 very strong). Moreover, when hovering simultaneously on different arcs, all their descriptions are merged in a single tooltip.

#### 6.3.2.2 Expansion

If we click right on a class displayed in KC-Viz, in this case *Work*, we obtain a menu that includes options for inspecting, expanding, and hiding a class. If we select "Expand", a menu pops up, which provides a rich set of options for exploring the subtree under class *Work*, as shown in Fig. 6.7. In particular, the following four options for customising the expansion algorithm are presented to the user:

- whether to explore following taxonomic relations, other relations (through domain and range), or any combination of these;
- whether or not to make use of the KCE algorithm, which in this case will be applied only to the subtree of class *Work*;
- whether or not to limit the range of the expansion—e.g., by expanding only to one or two levels;

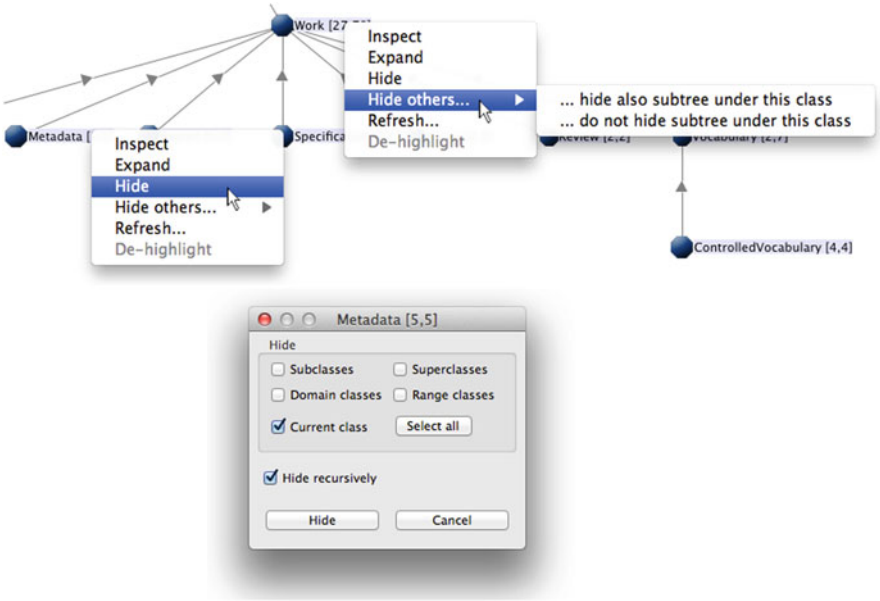


**Fig. 6.8** The two options for hiding concepts: "Hide", applied on the class *Metadata*, and "Hide others . . . " used on the class *Work*

- whether to display the resulting visualisation in a new window ("Hide"), or whether to add the resulting nodes to the current windows[36].

#### 6.3.2.3 Hiding

After right-clicking on a node, two options, "Hide" and "Hide others" implement a flexible mechanism for hiding nodes, as shown in Fig. 6.8 on page 220. If we select "Hide", a menu pops up, which provides a rich set of options for hiding the class selected and its subtree. By selecting "Hide others", one can choose to hide all the classes but the selected one and, optionally, its subclasses.

[36] In the latter case, some degree of control is given to the users with respect to the redrawing algorithm, by allowing them to decide whether or not to limit the freedom of the graph layout algorithm to rearrange existing nodes. This is particularly useful in those situations where the expansion is meant to add only a few nodes, and the users do not want the layout to be unnecessarily modified—e.g., because they have already manually rearranged the nodes according to their own preferences.

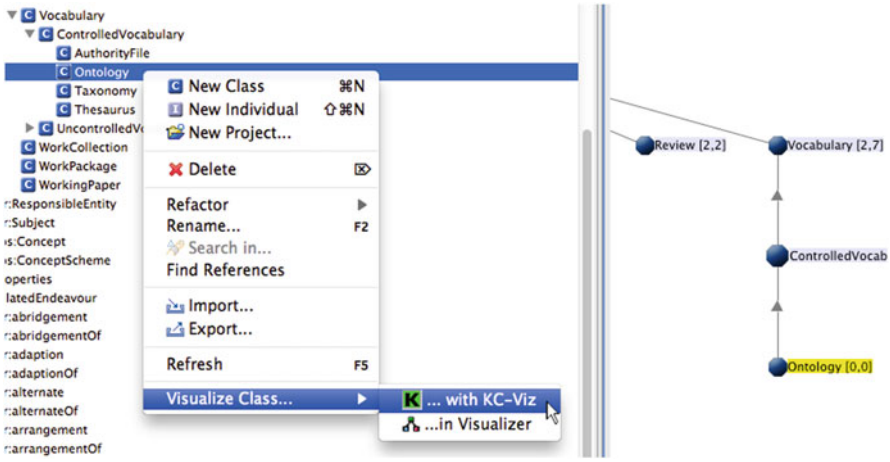


**Fig. 6.9** The option "Visualize Class with KC-Viz" to highlight (and eventually add) the class in the current KC-Viz panel

#### 6.3.2.4 Refresh Visualisation

The pop-up obtained after right-clicking inside the KC-Viz view has a new option called "Refresh", which allows the user to re-sync the visualisation with respect to changes in the model, which may have occurred since the visualisation was produced.

#### 6.3.2.5 Integration with NeOn

KC-Viz is integrated within the core components of the NeOn Toolkit, including the Entity Properties View and Ontology Navigator. This means that it is possible to click on nodes in KC-Viz and highlight them in these components (option "Inspect"), as well as click on items shown in the Ontology Navigator and adding them to the visualisation in KC-Viz, as shown in Fig. 6.9.

#### 6.3.2.6 Dashboard

A dashboard containing buttons for acting on the current KC-Viz panel is positioned immediately above the visualisation panel, as shown in Fig. 6.10 on page 222.

In the dashboard there are history buttons that allow the user to move back and forth through the history of KC-Viz operations. Each operation, or move, can be distinguished in macro (extraction and hiding) and micro (re-layout, axis adjustment, node movement) moves. It is important to notice that micro moves can be cancelled/re-applied only if they are not preceded/followed by macro moves in the operation history. For instance, let us consider the sequence "Mac1 mic1 mic2 Mac2 mic3 . . . " and let us suppose to be visualising in KC-Viz the window after mic2 and before Mac2. A user can decide:

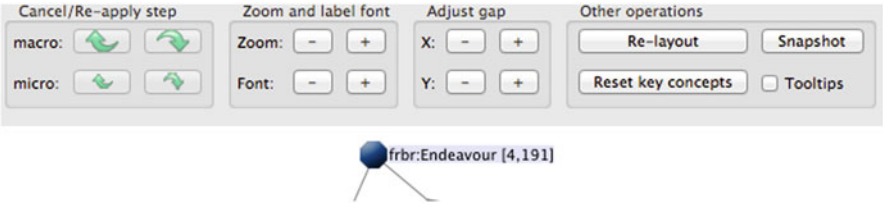


**Fig. 6.10** The dashboard which allows the user to move back and forth through the history of KC-Viz operations, to modify the formatting of the layout, and to save the current display to a file, among other things

- to cancel mic2;
- to cancel Mac1;
- to re-apply Mac2.

Moreover, KC-Viz provides an essential interface customisation support, such as graphical zooming, font manipulation and tree layout customisation. For the latter, after clicking on the button "Re-layout" in the KC-Viz toolbar, a sub-menu is opened, which asks the users whether they want to redraw the current visualisation using a top-bottom orientation or a left-right orientation.

Finally, through the button "Snapshot" inside the dashboard, it is possible to save the current display to a file or to load/delete a previously stored one.

#### 6.3.2.7 Preferences

KC-Viz has a preferences panel, shown in Fig. 6.11, which allows the user to set defaults for the most common operations and also enables them to switch to a more efficient (but sub-optimal[37]) algorithm when dealing with very large ontologies. Moreover, all the weights of the formulae used in the KCE algorithm are customisable by the user. In particular, the preference panel allows users to change the relative weights of the different criteria used to calculate the overall score for each class of an ontology. Customisation options are also provided for all the criteria.

### *6.3.3 Empirical Evaluation*

In order to gather initial data about the performance of KC-Viz, Motta et al. (2011a), introduce a preliminary empirical evaluation, which required 21 subjects to perform four ontology engineering tasks (max. 15 min/task), involving ontology exploration. The tasks given to the subjects are shown in Table 6.4. This set of tasks was designed to ensure coverage of different exploration strategies, which are typically required in the context of a sense-making activity.

[37] When this preference is enabled the set of key concepts returned by the algorithm may not guarantee the best possible *coverage* of the ontology.

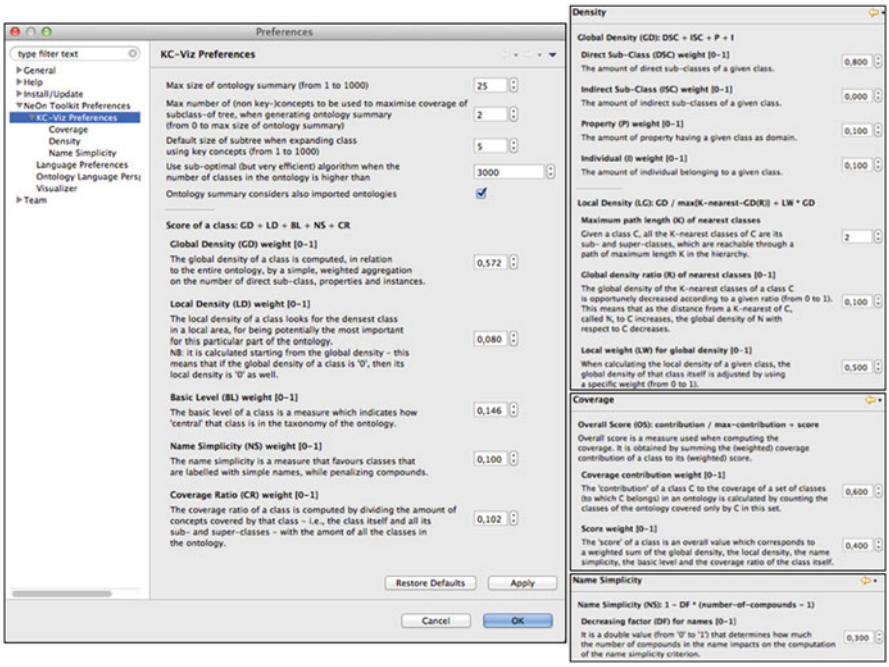


**Fig. 6.11** The preference panel of KC-Viz

**Table 6.4** Ontology engineering tasks

| | |
|---|---|
| *T1* | Which class has the highest number of direct subclasses in the ontology? |
| *T2* | What is the most developed (i.e., has the biggest subtree) subclass of class Quantity found in the ontology at a concrete level of granularity (i.e., do not consider abstract classes which have the term 'quantity' in their id)? |
| *T3* | Find three subclasses of Agent, at the most abstract level possible (under Agent of course), which are situated at the same level in the hierarchy as each other, and are also subclasses of CorpuscularObject |
| *T4* | We have two individual entities (a particular copy of the book War & Peace and a particular 5 p coin). Find the most specific classes in the ontology, to which they belong, say P1 and P2, and then identify the most specific class in the ontology, say C1, which is a superclass of both P1 and P2—i.e., the lowest common superclass of both P1 and P2 |

The 21 subjects were randomly allocated to three different groups, labelled A, B, and C, where each group used a particular conuration of ontology engineering tools. In particular, members of group A carried out the tasks using the NeOn Toolkit v2.5, without any visualisation support, and they were only allowed to use the search functionality, the *Ontology Navigator* and the *Entity Properties View*. The role of this group was to provide a baseline to the experiment, providing us with some data on how effectively people can tackle ontology exploration tasks, without any visualisation support.

The members of Group C were asked to solve the tasks using KC-Viz, together with the search functionality provided by the NeOn Toolkit. To ensure a separation between groups A and C, members of the latter group were explicitly forbidden from using the Ontology Navigator for exploration, although they were allowed to use it as an interface between the search facility in the NeOn Toolkit and KC-Viz.

Finally, the members of Group B carried out the tasks using the Protégé 4 environment, v4.1.0, in particular using the search functionality, the class browser and the *OwlViz* plugin. This conuration was chosen for three reasons:

1. to compare KC-Viz to a robust tool, widely used in concrete projects by members of the ontology engineering community, in order to maximise the value of the experiment to the community;
2. while OwlViz uses the same node-link paradigm as KC-Viz, its design is rather different from KC-Viz; and
3. having considered the visualizers available in other state-of-the-art ontology engineering tools, such as the NeOn Toolkit (*Kaon Visualizer*) and TopBraid (*Graph View*), OwlViz appears to provide a more user friendly and flexible functionality than the comparable ones available in TopBraid and the NeOn Toolkit.

Before the experiment, each of the subjects filled a questionnaire, answering questions about their expertise in ontology engineering, knowledge representation languages, and with various ontology engineering tools, including (but not limited to) NeOn and Protégé. At the end of the experiment, every subject filled free text questions on the post-task questionnaire to provide views on the perceived strengths and weaknesses of the tool used during the experiment. Additionally, subjects who did not use KC-Viz provided feedback following a demo. Note that none of the subjects had much direct experience with the ontology used for the tasks (i.e., the SUMO ontology[38]).

Out of 84 tasks in total (4 times 21), 71 were completed within the 15 min time limit, while 13 tasks were not completed, a 15.47 % percentage failure. The 13 failures were distributed as follows: 5 in group A (NTK), 6 in group B (OwlViz), and 2 in group C (KC-Viz).

Figure 6.12 shows the average time (in seconds) taken by each group in each task, as well as the total averages across groups and tasks[39]. As shown in the table, on each of the four tasks the fastest mean performance was with KC-Viz, whose overall mean performance was about 13 min faster than OwlViz, which in turn was about two minutes faster than NTK. Although not significant, the difference in total time taken across the four tasks with the three different tools appeared to be approaching significance, $F(2, 20) = 2.655$, $p = 0.098$.

[38] The SUMO ontology: http://www.ontologyportal.org/SUMO.owl.

[39] For tasks not completed within the time limit, we consider a 15 min performance. This could be modified to consider 'penalties', such as a 5 min penalty for a non-completed task. However, adding the penalty does not lead to meaningful changes in the interpretation of the data, other than increasing the performance gap between the KC-Viz group and the others.

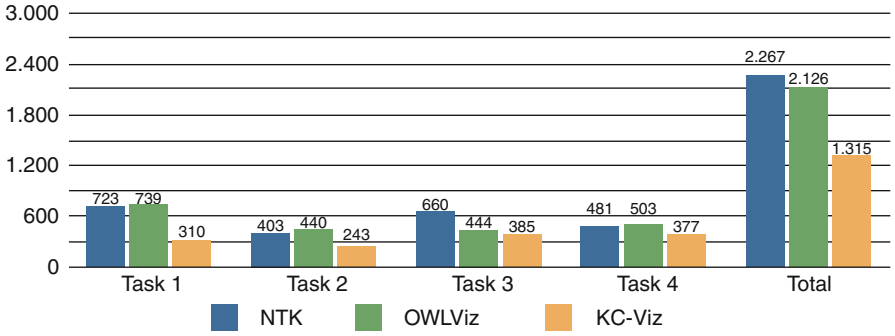


**Fig. 6.12** Performances (in seconds) for each task

**Table 6.5** Usability scores

| NTK mean | OWLViz mean | KC-Viz mean |
|---|---|---|
| 26.9 | 25.7 | 27.1 |

The difference in performance across the three tools on Task 1, was statistically significant $F(2, 20) = 9.568$, $p < 0.01$. A Tukey HSD pairwise comparison revealed a significant difference between both KC-Viz and NTK ($p < 0.01$) and KC-Viz and OwlViz ($p < 0.01$). Although mean performance was faster for KC-Viz across the board, performance differences on the other three tasks did not reach statistical significance. Nevertheless, these results suggest advantages for KC-Viz in supporting users in such realistic browsing and visualisation tasks.

From the questionnaire data, usability scores were calculated using the SUS formula (Brooke 1996) for each of the three conditions—see Table 6.5. The mean usability score was slightly higher for KC-Viz, though very similar across the three tools and not statistically significant.

The free text questions on the post-task questionnaire were analysed using a grounded theory approach (Birks and Mills 2011). This approach was used to build categories of comments that either expressed positive feedback, offered criticism, or suggested improvements. A category was discarded when it only contained comments from a single subject.

The three main categories of positive comments concerned:

- the flexible support provided by KC-Viz to manipulate the visual displays;
- the abstraction power enabled by the KCE algorithm;
- the value of the subtree summaries provided by KC-Viz.

These results are encouraging in the sense that they provide some initial indication that there is probably a direct causal link between the use of key concepts as an abstraction mechanism and the good performance of KC-Viz on the evaluation tasks, even though these were not designed specifically to map directly to KC-Viz features.

Of the three main categories of negative comments described in Motta et al. (2011a), only one still remains unsolved in the current version of KC-Viz: the lack

of integration between KC-Viz and reasoning/query support in the NeOn Toolkit. Obviously, future versions of the tool will be opportunely extended to address this and other minor issues.

## 6.4 Graffoo, a Framework for Visual Ontology Modelling

Twenty years ago the only way Web users had to publish Web pages was to write HTML documents through text editors and, then, uploading them on the Web by means of appropriate transfer protocols (e.g., FTP). Obviously people might not be self-sufficient to accomplish this task. A first step forward was done in 1997 ca., when WYSIWYG editors for HTML documents were introduced (e.g., *Microsoft Frontpage*). However, although these editors solved the issue of writing HTML pages through the adoption of user-friendly interfaces, a problem still persisted: where and how to upload the documents.

The situation dramatically improved when the paradigm of creating and publishing HTML documents started to involve Web-based application rather than desktop applications. The introduction of blog platforms, wikis, CMSs and other social tools has generated an exponential increase of people writing for and publishing on the Web. The winning strategy has been characterised by two main points:

- to hide the complexity of the HTML markup behind Web interfaces;
- to reduce the entire publishing process to pressing a button.

Currently, some aspects of the Semantic Web, such as the creation of OWL ontologies and RDFS vocabularies, still continue to be discussed within specific research communities, e.g., the Semantic Web and the Ontology Engineering. This precludes a broad adoption of these technologies by non-expert Web users, since people cannot deal with some important semantic technologies unless having proper skills. Like the above-mentioned example about the creation and publication of HTML documents, Web users would start to write and publish OWL ontologies for/on the Web only when:

- there will exist proper interfaces that hide the intrinsic complexity of the OWL framework behind user-friendly GUI;
- the publication process of OWL ontologies will be automatised and reduced to the pressing of a button.

In this section I will describe a work that deals with the first of the above points. Similarly to what I presented in Sect. 6.1.3, here I will introduce the *Graphic framework for OWL ontologies*[40] (*Graffoo*). It is a tool that can be used to present OWL ontologies, or sub-parts of them, through clear and easy-to-understand diagrams.

All the objects that can be used in a Graffoo diagram are shown in Fig. 6.13. These have been developed using the standard library of *yEd*[41], a free diagram editor

[40] Graffoo, the Graphic framework for OWL ontologies: http://www.essepuntato.it/graffoo.

[41] The yEd diagram editor: http://www.yworks.com/en/products_yed_about.html.

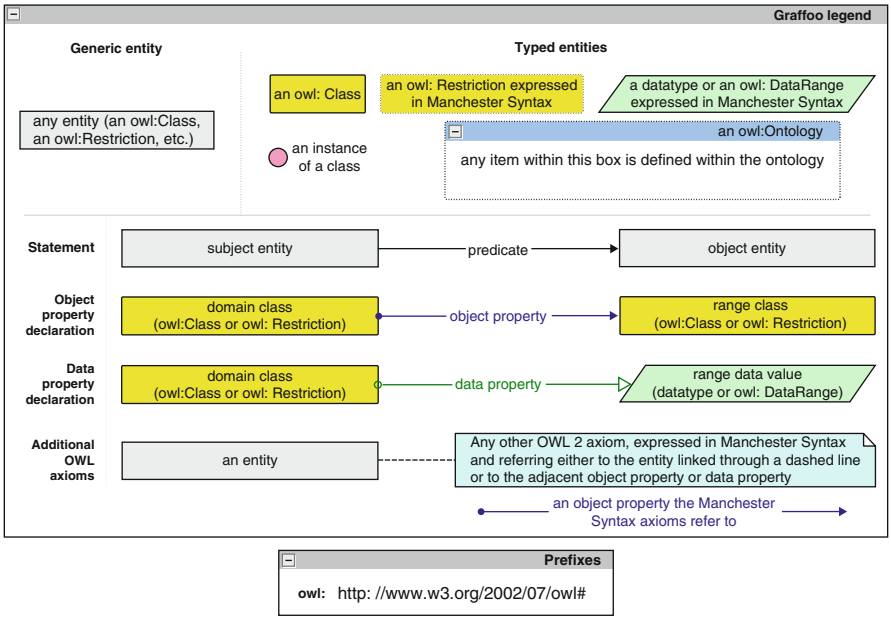


**Fig. 6.13** The legend for all possible Graffoo objects

running on Windows, Mac and Linux. The *graphml* format version of these objects is also available[42] and can be loaded as proper section in the yEd palette.

From some preliminary informal studies, it seems that the Graffoo representations of OWL ontologies can be comprehended without the need to understand all the details of OWL 2 or of any of its linearisations (Turtle, RDF/XML, Manchester Syntax, OWL/XML). In particular, when I was developing the SPAR ontologies, I used Graffoo when I needed to illustrate an ontology to users who were not expert in Semantic Web technologies, but who were interested in understanding SPAR ontologies.

In the following sections, I will illustrate how to use Graffoo widgets to formalise ontologies. As example, I use some excerpts from the EARMARK ontology introduced in Sect. 3.1.

### *6.4.1 Introducing Classes and Properties*

In Graffoo all the entities can be defined either as an URI surrounded by angular brackets or as a CURIE with a prefix. All the prefixes can be defined within a particular box (entitled "Prefixes") as a list of prefix-URI couples.

[42] http://www.essepuntato.it/graffoo/sources.

The classes are drawn as yellow rectangles, while green rhomboids describe datatypes, and both widgets have solid black borders. Object and data properties are created linking classes, restrictions (introduced in the following section) and datatype through particular arrows. The object property declarations are defined through blue solid lines, where the solid circle at the beginning identifies the domain while the solid arrow at the end indicates the range. The data property declarations are drawn as green solid lines, where the empty circle at the beginning identifies the domain while the empty arrow at the end indicates the range. Moreover, it is possible to associate additional axioms to properties putting a light-blue box close to (or upon) the property it refers to.

For instance, the following excerpt is the Manchester Syntax linearisation of the diagram in Fig. 6.14 on page 229:

```
Prefix: earmark: <http://www.essepuntato.it/2008/12/earmark#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>

Class: earmark:Range

Class: earmark:Docuverse

ObjectProperty: earmark:refersTo
  Domain: earmark:Range
  Range: earmark:Docuverse
  Characteristics: Functional

DataProperty: earmark:begins
  Domain: earmark:Range
  Range: rdfs:Literal

DataProperty: earmark:ends
  Domain: earmark:Range
  Range: rdfs:Literal

DataProperty: earmark:hasContent
  Domain: earmark:Docuverse
  Range: rdfs:Literal
  Characteristics: Functional
```

### 6.4.2 *Defining Restrictions and Additional Class Axioms*

The widget that defines restrictions is a light-yellow rectangle with dotted border. Classes and restrictions can be linked to create assertions (e.g., *rdfs:subClassOf* relations) through black solid arrows, labelled according to the property involved in

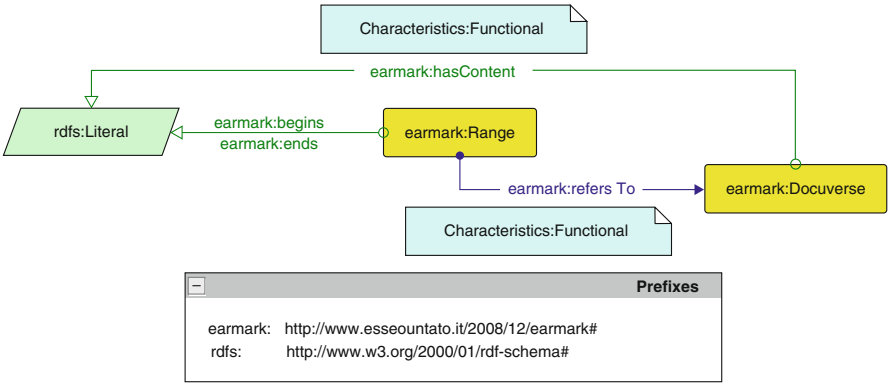


**Fig. 6.14** Widgets defining prefixes, classes, object/data properties and property axioms

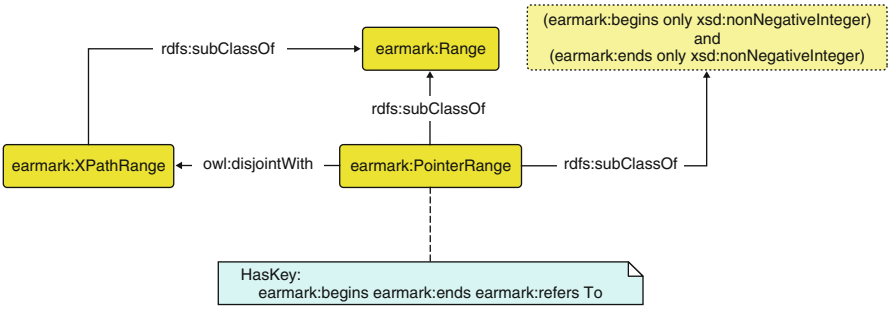


**Fig. 6.15** Widgets defining restrictions and other class axioms

the assertion in consideration. Moreover, additional axioms (in Manchester Syntax) can be associated to classes using a light-blue box linked through a dashed line.

For instance, the following excerpt is the Manchester Syntax linearisation of the diagram in Fig. 6.15:

```
Class: earmark:PointerRange
  SubClassOf:

    earmark:Range ,
    (earmark:begins only xsd:nonNegativeInteger) and
      (earmark:ends only xsd:nonNegativeInteger)
  DisjointWith: earmark:XPathRange
  HasKey: earmark:begins earmark:ends earmark:refersTo

Class: earmark:XPathRange
  SubClassOf: earmark:Range
```

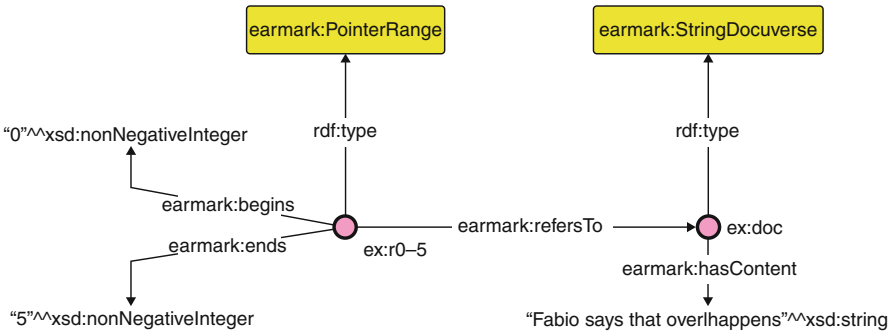


**Fig. 6.16** Widgets defining individuals and related assertions

### 6.4.3 *Linking Class Individuals*

Graffoo allows one to define also ABox of ontologies, by creating individuals having a particular type. The widget used to define those individuals is a pink circle with solid black border. Note that this widget specifies the URL/CURIE of the related individual around it, rather than in the middle of it. As all the other resources, individuals can be linked to create assertions through (labelled) black solid arrows.

For instance, the following excerpt is the Manchester Syntax linearisation of the diagram in Fig. 6.16:

```
NamedIndividual: ex:r0-5
  Types: earmark:PointerRange
  Facts:
    earmark:begins "0"^^xsd:nonNegativeInteger ,
    earmark:ends "5"^^xsd:nonNegativeInteger ,
    earmark:refersTo ex:doc

NamedIndividual: ex:doc
  Types: earmark:StringDocuverse
  Facts:
    earmark:hasContent "Fabio says that overlappens"^^xsd:string
```

### 6.4.4 *Defining Assertions Between Ontologies*

Graffoo includes a specific widget—a transparent box with a light-blue heading and a dotted black border—for the specification of ontologies. The ontology URIs are put in the heading of this widget.

All the entities contained by this widget are formally defined within the related ontology. Moreover, like the other resources, ontologies can be linked to create assertions (e.g., *owl:imports*) through (labelled) black solid arrows.

For instance, the following excerpt is the Manchester Syntax linearisation of the diagram in Fig. 6.17:

```
Ontology: <http://www.essepuntato.it/2008/12/earmark>
  Import: <http://www.essepuntato.it/2010/05/ghost>

Class: earmark:PointerRange
  SubClassOf:
    earmark:Range ,
    (earmark:begins only xsd:nonNegativeInteger) and
      (earmark:ends only xsd:nonNegativeInteger)
  DisjointWith: earmark:XPathRange
  HasKey: earmark:begins earmark:ends earmark:refersTo

Class: earmark:XPathRange
  SubClassOf: earmark:Range

Class: earmark:Range
  Annotations: rdfs:isDefinedBy
    <http://www.essepuntato.it/2010/05/ghost>

Class: earmark:Docuverse
  Annotations: rdfs:isDefinedBy
    <http://www.essepuntato.it/2010/05/ghost>

ObjectProperty: earmark:refersTo
  Annotations: rdfs:isDefinedBy
    <http://www.essepuntato.it/2010/05/ghost>

DataProperty: earmark:begins
  Annotations: rdfs:isDefinedBy
    <http://www.essepuntato.it/2010/05/ghost>

DataProperty: earmark:ends
  Annotations: rdfs:isDefinedBy
    <http://www.essepuntato.it/2010/05/ghost>
```

## 6.5 Gaffe, a Flexible and User-Friendly Authoring Tool for Semantic Data

The process of associating semantic data to resources, such as documents, is quite complex in principle. The first issue is that several and alternative models (e.g., metadata schemas, vocabularies, ontologies) can be used to describe the same resource within a particular domain (e.g., publishing). Some of these models are almost equivalent, others are characterised by individual features. For instance, all metadata models for the description of bibliographic documents are expected to include information about the "author", "publisher" and "year". At the same time, a schema describing Ph.D. theses needs to include similar information (for instance, "author"

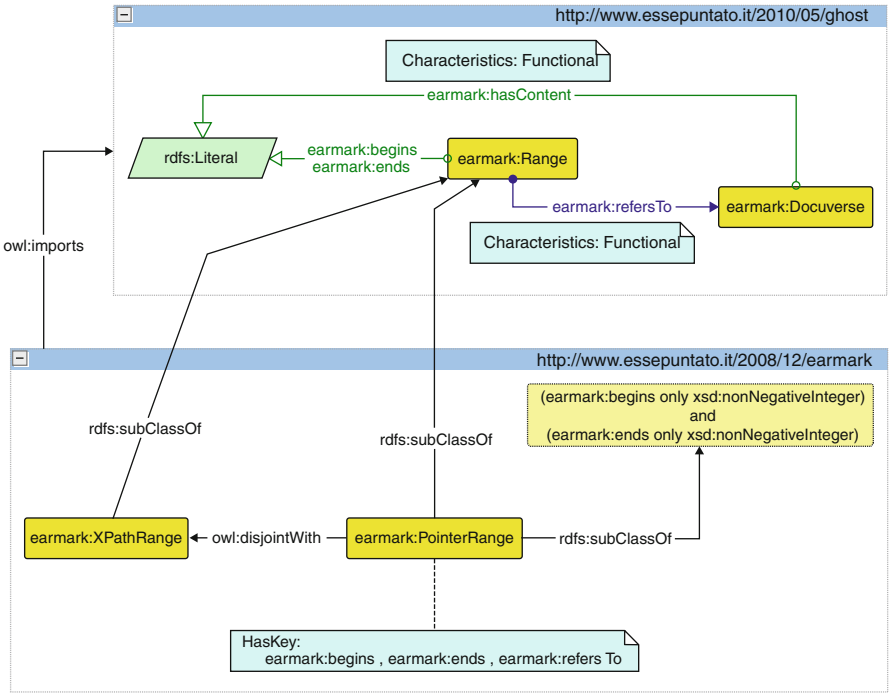


**Fig. 6.17** Widgets for defining ontologies and related assertions

and "year") and more domain-specific data such as "id number", "supervisor" and "discipline". The choice of the most suitable model depends on two main factors:

- the nature of the resource (e.g., a document);
- the applications (and users) that the resource is meant to be processed by (e.g., Wikis and word processors).

Still, choosing the appropriate model is not sufficient. It is also important that an interface enables the creation of semantic data in an intuitive and usable way, thus hiding the complexity of the particular formalism or model used. A good model that is not supported by a good editing interface risks to be useless: authors would often decide not to insert data, considering it a pointless, time-consuming and deferrable task.

I have identified four main features that a flexible and user-friendly authoring tool for semantic data should have:

- *genericness*, the editor should support any model in a flexible way;
- *customisability*, instead of generating a static form—strictly dependent on a given model—the editor should be customisable for models, users' needs and preferences;

- *proactivity*, the editor should provide users with facilities that simplify the authoring process, such as pre-filled form fields, suggestions, default values, access to environment variables, and so on;
- *validation*, the editor should apply validation mechanisms to check the correctness of the inserted values.

A solid approach to flexible interfaces consists in adopting the "Model-View-Controller" (MVC) pattern (Gamma et al. 1994), as developed in the software engineering community. This pattern implements a clear separation between the *business logic* of an application and the *user interface* for visualising/editing data: it allows designers to generate applications whose interfaces can be easily modified without affecting the model and vice versa.

Discussing the benefits of MVC is out of the scope of this book, but it is important to explore how this pattern can help to design a flexible and sophisticated metadata editor. In the context of metadata editors, the three components of MVC become:

- *Model* The model corresponds to the actual semantic data as manipulated by the editor and associated to the document/resource. Changing the model describing semantic data means changing the model of MVC, and this should always be possible, in order to obtain a model-independent editor;
- *View* The view is the way semantic data are shown to the users. Usually, the view can be classified according to two different types: the *edit interface* and the *visualisation interface*. The edit interface has to be a rich graphical interface, with a large number of graphical widgets to specify semantic data values according to their type and expected values. The more widgets are sophisticated and well-structured, the more easily users can create semantic data. The visualisation interface shows the current resource-related data only, without changing the internal model. The visualisation can happen through deactivated form fields, but also with plain textual or tabular visualisation. Since model and view are separated, it is possible to assign multiple views to the same semantic model, each tailored to roles, personal preferences and local policies of the intended users;
- *Controller* The controller is the component in charge of managing the interaction between the user and the application. It has to store the values provided by user into the model. Moreover, it is expected to run a *pre-processing phase* to provide default values to relevant form fields and a *post-processing* one to validate metadata values as provided by the user. The controller thus handles all input events and notifies the model of each user's action that generates changes to the model itself.

Of course, previous solutions that combined the MVC pattern with other approaches were had been proposed to address the issue of associating semantic data to electronic documents. In particular, my research group has been working for years on an approach consisting of two steps. First, creating ontological descriptions of the domain and of the interfaces to manipulate semantic data. Second, transforming those descriptions into the actual interfaces that are shown to the users. My personal contribution in this field is the development of the new Java implementation of *Gaffe* (*Generator of Advanced Forms and Friendly Editor*; Bolognini et al. 2009). Gaffe is a MVC-based API that makes it possible to build customisable editors for semantic

data, so as to allow users to annotate a resource according to any scheme as expressed through an OWL ontology. More precisely, Gaffe uses two different ontologies:

- the *domain* ontology represents the conceptual model. Since this ontology is unconstrained, users can adopt any custom model without any restriction, as long as it is expressible in OWL;
- the *GUI* ontology specifies the classes and properties of widgets and form elements of a graphical user interfaces, as well as the mapping between interface widgets and properties of the domain ontology.

The *instance* document is an instantiation of the GUI ontology to describe an actual interface, as generated by associating domain elements to graphical widgets and by customising the final appearance of each item.

In the following sections I will introduce a Gaffe-based prototypical application called OWiki, developed specifically for demonstrating the capabilities of the Gaffe principles.

### 6.5.1 OWiki: Ontology-Driven Generation of Templates and Forms for Semantic Wikis

OWiki (Di Iorio et al. 2010a, b, 2011a) is a Gaffe-based extension of MediaWiki that supports users in creating and editing semantic data. The basic idea of OWiki is to exploit ontologies and MediaWiki editing/viewing facilities to simplify the process of authoring semantic wiki content.

In particular, OWiki exploits MediaWiki templates, infoboxes and forms. A *template* is a set of pair *key-value,* edited as a record and usually formatted as a table in the final wiki page. Templates are particularly useful to store structured information, as they are: very easy to edit; disconnected from the final formatting of a page; very easy to search, and so on. Templates are defined in special pages that can be referenced from other pages. These pages include fragments with the same structure of the template but filled with instance data. The template-based component of a page is also called *infobox*.

OWiki exploits ontologies to represent the (semantic) knowledge base of a wiki and templates to display and add ABox assertions of the ontology through the wiki itself. The integration and interaction between ontologies and templates can be summarised in two points:

- each class of the ontology is associated to a template-page. Each property is mapped into a key of the infobox;
- each instance of that class is represented by a page associated to that template. Each line in the infobox then contains the value of a property for that instance. Data properties are displayed as simple text while object properties are displayed as links to other pages (representing other instances of the ontology).

OWiki templates are actually invisible to users. Each template is associated to a form that allows users to create and edit the relative instances. Users do not modify the templates *directly* but they only access specialised form fields.

The crucial point is that even forms are generated automatically from ontological data. Obviously, OWiki includes a *GUI ontology* describing widgets and interface elements. The concepts and relations of the *domain ontology*—that is the ontology according to which semantic data are specified—are mapped into form elements that are delivered to the final user.

During the installation phase, OWiki creates a basic set of forms by merging the domain ontology with the GUI one. At the editing phase, the system shows a very basic form and saves it as a special page (template). This page can then be organised as a new form by adding dynamic behaviours, moving buttons, changing the field order and so on.

Before describing the internal architecture of the system, it is worth spending few more words about the way OWiki uses ontologies. The extensive usage of ontologies makes it possible (1) to make OWiki independent on the domain it is used for, (2) to customise forms and templates easily, and (3) to describe the evolution of a wiki page and its semantic content fully.

#### 6.5.1.1 The Architecture of OWiki

OWiki is an integrated framework composed of three modules, delivered with different technologies:

- a *MediaWiki extension* It is a module integrated in MediaWiki written in PHP that adds OWiki facilities;
- an *Ontology manager* It is a Java web service that processes OWL ontologies to produce forms for editing metadata. This manager uses both Jena API (Carroll et al. 2004) and OWLAPI (Horridge and Bechhofer 2011);
- an *Ajax-based interface* A client-side module that allows users to insert data directly through the forms generated by the OWiki engine.

The PHP OWiki module follows the same architecture of any MediaWiki extension: some scripts and methods are overridden to provide new features. In particular, the module implements a revised editor that initialises the OWiki environment variables, sets the communication with the client and sets the data necessary to store forms in the MediaWiki database without interfering with existing data.

To manipulate ontologies, OWiki implements a web service that uses the Jena API. Jena is integrated with the Pellet reasoner (Sirin et al. 2007), which is exploited to extract information about the instances in the ontology. Ranges of some properties, as well as their values, are in fact derived from subsumptions or other relations expressed in the ontology itself.

The web-service specifically generates templates from the ontological data, which are later sent to the PHP module and stored in the right place of the MediaWiki installation.

The connection between the modules—PHP and Java—and the core of the overall framework is the OWiki client. The client is a javascript application, based on

Mootools[43], in charge of generating and delivering forms. It is strongly based on the Model-View-Controller (MVC) pattern and its internal architecture can be divided in four layers:

- *The Connection Layer* manages the overall environment, the initialisation phase and the communication between all other layers.
- *The Model Layer* (Model of MVC) manages the data to be displayed on the page. It is composed of a factory that creates wrappers for each type of data and instantiates data from the ontology.
- *The Look And Feel* (View of MVC) manages the final representation of the form, containing atomic and complex widgets, manipulators and decorators.
- *The Interaction Layer* (Controller of MVC) implements the logic of the application, the communication with the web-service, the generation of semantic data and the end-user interaction.

#### 6.5.1.2 Using Ontologies to Model the Domain

Since it implements the Gaffe architecture, in OWiki the entire *domain of discourse*—i.e., all the topics presented on each page—is handled by using a *domain ontology*. Two different kinds of classes exist in this ontology: those—*page-domain* classes—that strictly relate to articles and pages visualised by the wiki, and those—*data-domain* classes—that define additional data around the former ones.

Each individual of the *page-domain* classes results in a wiki page containing text content (the content is now stored in the MediaWiki internal database) and all the semantic data directly related to that individual. Figure 6.18 shows a page (about a particular type of beer[44]) that contains a textual description of it in the central page area, while in the right box there are all the metadata related to the beer.

While some metadata such as "Beer Alcoholic content" or "Beer Brewed by" are proper to any beer directly, since they are defined by OWL data or object properties having the class *Beer* as domain, that is not true for other metadata, such as "Winner Award" and "Winner Awarded on". In fact, those properties are handled using the *data-domain* class *Awarding* that represents an event concerning a particular prize occurred at a specific time. The model of such properties for the beer in consideration, shown in Fig. 6.18, is explained in the following excerpt (in Turtle syntax):

```
:carlsberg a :Beer ; :hasAwarding :awardingEuropean2007.


:awardingEuropean2007 a :Awarding ;
  :hasAward :europeanBeerAward ;
  :hasYear "2007" .
```

[43] Mootools: http://mootools.net.

[44] The demo installation of OWiki I use in the following examples is available at http://owiki.web.cs.unibo.it.

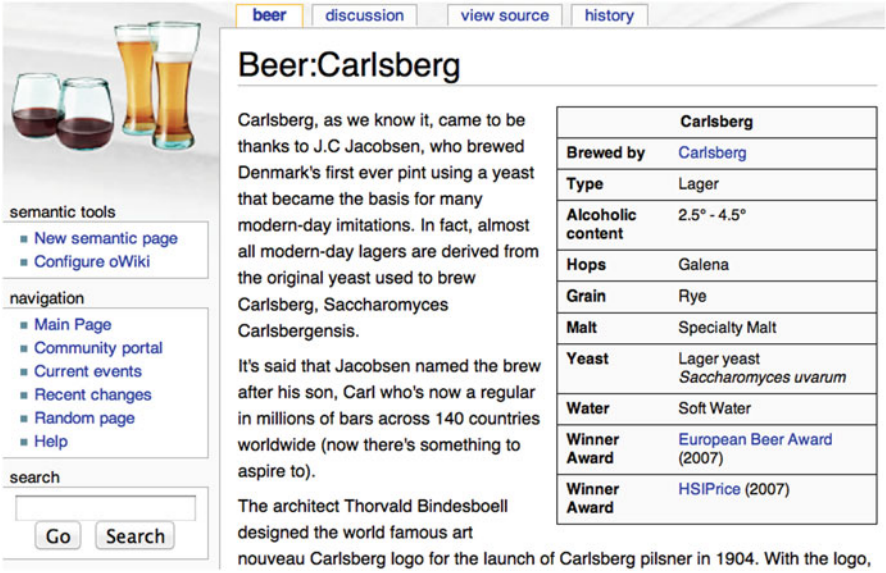


**Fig. 6.18** An example page of the Beer OWiki

The values shown in the Carlsberg page are not directly extracted from the Carlsberg ontological individual: they are taken from the awarding event the Carlsberg beer participated to.

Even if they are not directly represented as wiki pages, OWiki uses *data-domain* individuals to enrich even more the metadata of *page-domain* individuals. This enrichment is achieved by retrieving related data from non-page-domain individuals making at least two steps on the RDF graph represented by the model. We call *property flattening* the visualisation of those *data-domain* property values into a *page-domain* individual.

#### 6.5.1.3 Using Ontologies to Model the Interface

OWiki also exploits ontologies to model end-user interfaces. In particular, the system includes a preliminary version of the GUI ontology developed for Gaffe. This ontology is used for identifying all the components of web forms. The system instantiates and merges that ontology with the domain ontology in order to generate the final forms. The definitive version of the Gaffe GUI ontology is still under development but the core concepts and relations are stable and already tested in the current prototype.

The aim of separating the GUI ontology from the domain ontology is two-fold:

1. generating a declarative description of the interface widgets that can be reused across multiple domains and not being bound to specific data;
2. allowing users to customise the final interfaces by only changing the association between content and interface widgets.

It is to be noted that the GUI ontology can be designed once and for all, while the domain ontology requires different expertise for different application scenarios. The GUI ontology defines two types of graphical elements: *controllers* and *panels*. Panels are containers for other elements (that can be panels, in turn) used to organise the overall interface, while controllers are single widgets allowing users to fill metadata.

The main class of the ontology is *OWikiForm.* Instances of this class can be used to generate each form associated to each wiki page. Each instance in fact contains either graphical elements or property values from the domain ontology. *OWikiForms* can contain *simple* or *complex* types of controllers. Simple types are associated to data properties in the domain ontology, while complex types are associated to object properties.

Simple types model the basic form elements (*Textfield*, *ComboBox*, *CheckBox* e *RadioButton*) while complex types model constructs useful for mapping the GUI ontology to the domain one. There are two complex types: *ConnectField* and *Object-Container*. *ConnectField* model links to another wiki document. This is ultimately used to provide users with auto-completion operations on corresponding form fields: when the user fills this field, the system suggests a set of linked documents they can choose from (or create a link to a completely new resource). These links are in fact derived from the relations in the domain input ontology. *ObjectContainers* are widgets that include properties of a class which are not directly linked to the one defining a particular page, including to documents data about other (related) subjects. This class implements what I described earlier as *property flattening*.

## *6.5.2 Studying OWiki Through a Use-Case*

The main goal of OWiki is to simplify the creation of semantic data *through and within* wikis. The complexity of such metadata authoring process, in fact, is hidden behind the application so that users do not need to learn new interfaces and tools. Users can therefore easily create semantic data by exploiting forms and templates that are automatically generated from ontological data.

In this section I will explain in details this generation process, making clear how ontological data are converted into (customised) interfaces. Basically, the overall OWiki process consists of three steps:

1. ontology import and forms generation;
2. forms customisation;
3. templates and data generation.

### 6.5.2.1 From Ontologies to Forms

The first step consists of importing the input domain ontology into the wiki. Let us consider a sample application, which we will discuss throughout the following sections: an OWiki demo installation describing beers; breweries; ingredients, etc.

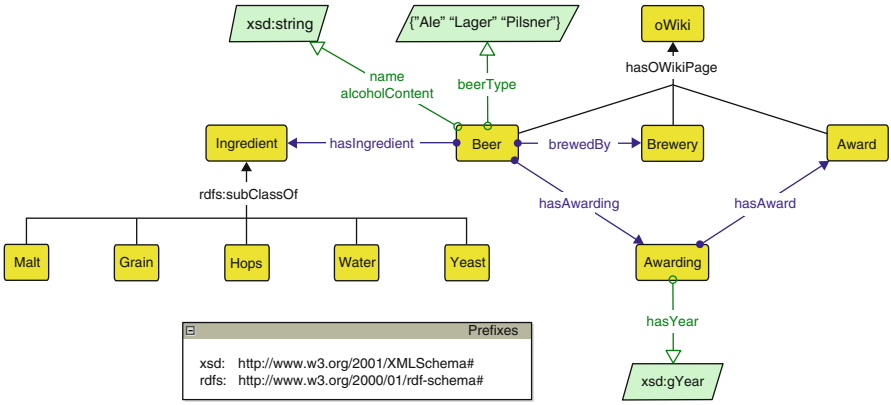


**Fig. 6.19** A graphical representation of the OWiki domain ontology about beers

Figure 6.19 shows some classes and properties of a domain ontology suitable for such an application.

Classes and properties are mapped into wiki pages as follows: each concept is mapped into a page and properties are expressed through templates. In particular, data properties become lines of templates infoboxes and object properties become typed links.

Note that the overall status of the OWiki installation is consistent at this stage, assuming that the domain input ontology was consistent. The process is in fact a straightforward translation of classes and relations into pages and links.

The OWiki conversion process also produces forms to edit the ontological content. Forms are dynamically built by analysing the class properties of the imported ontology and by mapping each property in the proper element of the GUI interface.

In the example, the class *Beer* defines three properties: *name*, *beerType* and *alcoholContent*. According to the type of these properties OWiki generates text fields or radio buttons; the default element is a text field that allows any type of value. Since in the input ontology the only possible values of the property *beerType* are "Ale", "Lager" and "Pilsner", the system add to the form a RadioButton element specifying those values.

For object properties OWiki chooses between two types of widgets according to their range: if the range class is associated (through the property *hasOWikiPage*) to the class *oWiki*, the system adds a *ConnectField* to the form; otherwise it adds an *ObjectContainer*. Since the class *Beer* has the object property *brewedBy* with the class *Brewery* specified as range, the system adds to the form a widget that allows one to include a link to a corresponding brewery page. This widget will also provide auto-completion features built on top of the relations expressed in the input ontology.

It is very important to make it clear that: there is a default mapping between classes of the domain ontology and elements in the GUI ontology based on *the type of the properties*. The name of a property or its meaning in a specific domain is not relevant.

There is indeed a conuration file that specifies, for each type, which widget to use and how to conure it. In the previous case, for instance, there was an association between enumerations and radio buttons. This mapping is deployed whenever a class has a property that may only have a finite set of values, regardless of the actual domain ontology. A change in the OWiki conuration file would be reflected by the use of a different widget for the same property.

#### 6.5.2.2 Forms Customisation and Filling

Furthermore, OWiki includes a conuration interface that allows users to set a domain-specific mapping between the input (domain and GUI) ontologies, and to conure the overall organisation of the form and its formatting properties.

The first time a user edits a page, OWiki shows a basic form. The author can then organise a new form adding dynamic behaviours, moving buttons, changing fields order and so on. Figure 6.20 shows a simple example of a customised form: while the original form only listed a set of plain text-fields, this one is organised in panels and uses radio-buttons, images and dynamic widgets.

Customisation can happen at different level. The user can change colour, font and background of the text to increase the appeal and impact of the form; they can change the position and the order of the elements to increase the importance of certain data; they can change the optionality of the elements, their default values, and so on.

The current implementation requires users to customise forms by editing an XML conuration file, through the wiki itself. Even if such an approach is not optimal, the internal architecture of the system relies on a strong distinction between the declarative description of the form (through the GUI ontology) and its actual delivery. That makes possible to implement a user-friendly and graphic environment to create and customise forms. A future development I will be working on is the implementation of such an editor within the OWiki framework.

#### 6.5.2.3 From Semantic Data to Templates and Views

Automatically-generated forms are finally exploited by the wiki users to write the semantic data. As described in the previous section, data are stored as templates and templates are manipulated by forms *in a transparent manner*.

Let me consider again the *Beer* class of the example. OWiki generates a form to create instances of those classes showing three main components:

- a text field to insert the name of the beer;
- a radio-button to select the type of the beer. Values in the radio button are directly extracted from the domain ontology;
- a text field to insert the brewery, which suggests breweries by exploiting information in the domain ontology.

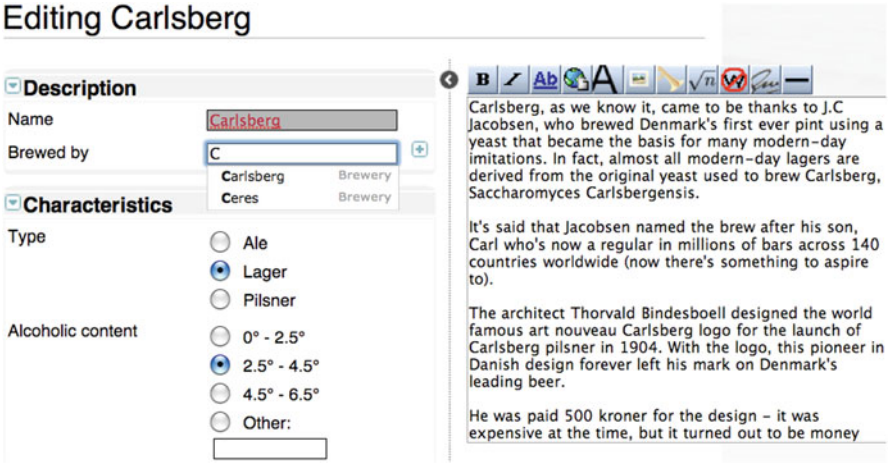


**Fig. 6.20** A customised form generated by OWiki

These components can even be organised in multiple panels. Once the user has filled the form OWiki saves a template with the proper information. Infobox templates, in fact, are used to display metadata and to cluster information about the same document.

Each infobox line corresponds to a field in the form that, in turn, corresponds to a parameter and its value in the domain ontology. As expected, the data properties of a class are displayed as simple text while the object properties are displayed as links to other documents.

The page corresponding to the Carlsberg beer in the example, that is an instance of the class *Beer*, and has been edited via the corresponding form, will contain the following (partial) infobox:

```
{{Infobox Beer |
hasoWikiNamePage=Carlsberg |
Beer_brewedBy=[[Brewery:Carlsberg|Carlsberg]] |
Beer_beerType=Lager |
Beer_hasAlcoholicContent=2.5 - 4.5 |
Hops_hasName=Galena | ... }}
```

Note that the property *Beer_brewedBy* contains a link to the page *Carlsberg* that is now an instance of the *Brewery* class. Relations in the input ontology are then mapped into the links between pages. The *Carlsberg* instance follows the same approach, being described by the infobox:

```
{{Infobox Brewery |
hasoWikiNamePage=Carlsberg |
Brewery_hasAddress=Valby 11 DK - 2500, Copenhagen |
Brewery_brews=[[Beer:Carlsberg|Carlsberg]] }}
```

Some final considerations are worth here about the consistency of OWiki. First of all, note that OWiki forms only work on the instances of the underlying ontology, without any impact on the classes and relations among them. The consequence is that, assuming that users do not corrupt infoboxes (that are anyway available in the

source code of a wiki page), the overall ontology keeps being consistent. The OWiki instance is in fact consistent *by construction* with the domain and the GUI ontology and it is populated via forms in a controlled way.

Thus, we can conclude—going back to the distinction between "wikis for ontologies" and "ontologies for wikis" proposed in Sect. 6.1.4—that OWiki currently belongs to the second group and does not properly use the wiki to build and update ontologies. In the future, I also plan to investigate a further integration between the wiki and the ontology—and a further integration between the textual content of a wiki page and the relative infoboxes—in order to use OWiki as a fully-fledged simplified authoring environment for ontologies as well.

## 6.6 From LIME to Semantically-Enhanced Legislative Editors

The legislative process is composed by two different elements: the *legal* aspect and the *documental* aspect. While the former concerns the application and evaluation of existing legislation within a particular scope, the latter, which is of particular interest for the topics of this book, addresses issues related to the documentary organisation of legislative texts, in terms of both the content (from the first draft to subsequent revisions) and the formats recording that content (from analogue to digital ones).

As introduced in Sect. 2.2, several markup languages have been developed in the past to provide robust mechanisms to define legal and legislative documents through standard digital formats such as XML. However, we still have to tackle the same point: user-friendly interfaces are needed to guarantee a broad adoption of those languages in specific contexts such as parliaments, commissions, governments, and so on. Tools enabling the creation of documents in those formats, such as editors and Web platform, are fundamental to promote the diffusion of such technologies.

One of the strategic advantages that have guaranteed the broad adoption of Akoma Ntoso (see Sects. 4.1 and 2.2.6) in real contexts is derived from the release of (standalone and Web-based) editors for Akoma Ntoso documents. These applications—namely Bungeni, AT4AM and LIME, which I will introduce in the following subsections—are thought to support domain users in structuring plain text through Akoma Ntoso markup vocabulary, even when the users are not aware of the particular technologies used to write and store legislative digital texts.

At the moment of writing this section (2013), the current implementation of such editors is quite robust for what concerns the application of XML-based markup structures upon plain text content. However, their future aim is also to enable the enhancement of documents through semantic annotations (i.e., RDF statements and OWL axioms) according to particular user-defined models, so as to let such editors evolve into fully-compliant Semantic Web applications.

In this section I will introduce briefly the aforementioned editors for Akoma Ntoso documents and then I will sketch out a list of requirements for the next generation of Semantic Web-aware legislative editors of the future[45].

### *6.6.1 Bungeni*

*Bungeni* [46] (English translation: *inside Parliament*) is a collaborative software development initiative based on Akoma Ntoso and other open source applications that provides a leading solution for drafting, managing, consolidating and publishing legislative and other parliamentary documents. It aims of making Parliaments more open and accessible to citizens.

One of the outputs of Bungeni is the development of an editor for legislative documents, called *Bungeni Editor*[47], which is a markup framework for the production of XML documents that conform to the Akoma Ntoso DTD. In particular, the Bungeni Editor was developed to manage a particular kind of Akoma Ntoso document, i.e., *reports*, which are documents containing any written account of parliamentary proceedings.

The editor is an extension of OpenOffice Writer[48], and allows users to enrich documents written through the OpenOffice Writer interface by means of three particular windows: the *control panel*; the *metadata editor*, and the *markup panel*, as shown in Fig. 6.21.

The control panel is used to create and open documents. It provides several functionalities such as text boxes which allow editors to add notes to the text; metadata visualisation; document validation against Akoma Ntoso DTD, and other tools to transform documents from the ODT format into others.

The metadata editor is used to enter preliminary and/or required metadata (the latter are accompanied by an "*" in the interface) to further mark the document up. In particular, the tab "General" contains fields to specify information related to the Akoma Ntoso document, while the other tabs (e.g., "Parliament" in the example in Fig. 6.21) are specific to particular kinds of document as they are defined in the "General" tab. In addition to defining metadata for the document level, it is also possible to specify information to non-overlapping blocks of text within the same document, since it may happen that a text involves particular agents as actors (e.g., the parliament, the judge, the language used, etc.) taking part in the event it describes (e.g., a speech).

[45] Note that it is difficult to freeze a moment for a complete description of a rapidly evolving situation related to ongoing projects and applications such as those described in the following sections. Therefore, all the considerations and descriptions about such tools refers to their status in June 2013.

[46] Bungeni project homepage: http://www.bungeni.org/.

[47] Bungeni editor homepage: https://code.google.com/p/bungeni-editor/.

[48] OpenOffice Writer homepage: http://www.openoffice.org/product/writer.html.

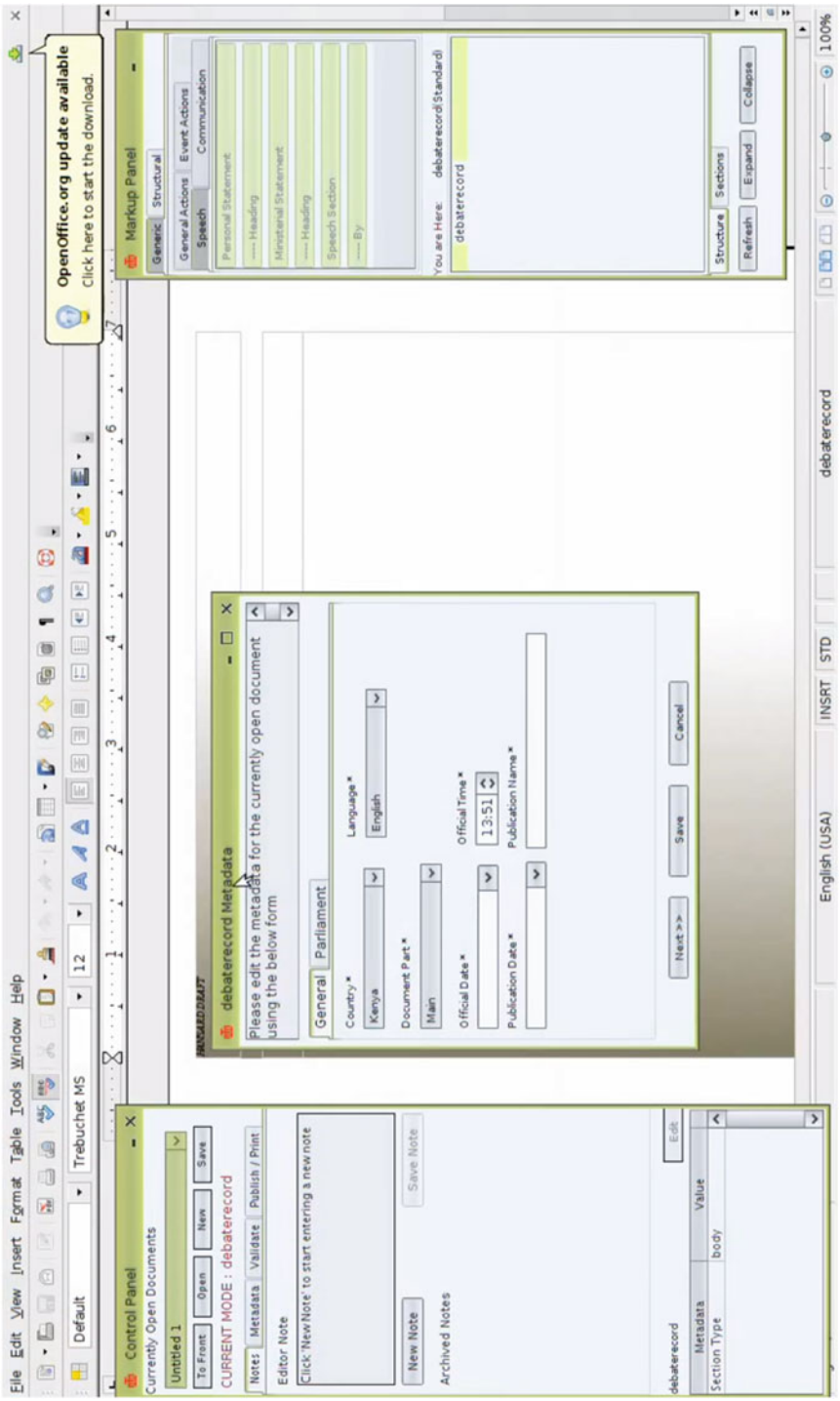


**Fig. 6.21** The three additional windows Bungeni made available to enrich documents defined as OpenOffice Writer documents

Finally, the markup panel allows users to add specific markup generics and structures to text, so as to identify speeches, sections, paragraphs, prefaces, and similar. All these markup items can be applied to the textual content of the document according to appropriate contexts; certain markup entities can be specified inside others while others cannot.

### 6.6.2 AT4AM

*AT4AM*[49] is as web-based authoring tool for amendments, broadly adopted within the European Parliament since 2010. This tool allows members of the Parliament to create and table (1) amendments on proposals coming from the European Commission and the Council of the European Union, and (2) reports of the parliamentary committees.

Recently, *AT4AM for All*[50] is the free and open source release of the AT4AM Web application, which was developed in Java and uses the Google Web Toolkit (GWT). Its authors praise it as the first system of the future XML-based workflow of amendments at the European Parliament entirely based on Akoma Ntoso to take as input and store amendment documents.

As shown in Fig. 6.22, the interface available to users to suggest amendments focuses only on the content of the proposal rather than the formal layout in which it will be finally formatted and released, and makes it easy to suggest modifications (link "Amend") to a particular recital and even create new (link "New") and delete existing structures (link "Delete") such as recitals, articles, paragraphs, sections, clauses, etc.

This system is currently used by the European Parliament, the European Commission, the United Nations and the Italian Senate.

### 6.6.3 LIME

The *Language Independent Markup Editor* (a.k.a. *LIME*)[51] (Palmirani et al. 2013) is a Web-based and multi-language editor for Akoma Ntoso documents. It takes as input a plain TXT file containing the content of the legislative document to mark up and provides a user-friendly interface that makes available several actions on the text.

At the moment of writing this section (2013), LIME is able to define bills and acts according to the Akoma Ntoso vocabulary. However, the developers are going to extend its functionalities so as to manage the whole Akoma Ntoso specification.

[49] AT4AM homepage: http://www.at4am.org/.

[50] A demo of AT4AM for All is available at http://www.at4am.org/demo/.

[51] LIME homepage: http://lime.cirsfid.unibo.it.


**Fig. 6.22** The AT4AM for All interface available to European Parliament members to suggest amendments to proposals

The LIME interface, shown in Fig. 6.23 on page 248, is organised into three main areas. On the left hand side, the "Outline" box shows the markup structures currently associated to the document content, presenting hierarchical containments as expandable nodes when such relations exist between markup elements. In the central part of the window there is the textual content of the document and a toolbar that makes available different actions, which are: editing the document content; previewing the document in PDF, and suggesting amendments to document text by means of the AT4AM for All service presented in Sect. 6.6.2. Finally, the right hand side of the interface provides a list of markup entities that can be used to mark up the document content, which conform to Akoma Ntoso.

LIME also provides tools to export the document in several formats (i.e., Akoma Ntoso XML, HTML, PDF and EPUB) and to infer some markup structures automatically, such as references and dates. In addition, every time a user try to add markup structures within the content, LIME performs a preliminary validation of the markup against the Akoma Ntoso DTD, so as that no markup can be added within wrong contexts. For instance, considering the text in Fig. 6.23, labelling the text from "Titulo II" to the end of the document as article is denied by the editor (since that text is already part of an article) and no markup will be added in this case. These kinds of checks are performed in conformance with Akoma Ntoso DTD and, in addition, with the meta-markup organisation defined through structural patterns (see Sect. 3.3.2), to which Akoma Ntoso adheres strictly, to define the content model of its markup elements.

### *6.6.4 Towards the Next Generation of Legislative Editors*

During the Extended Semantic Web Conference held in Montpellier in 2013, Prof. Enrico Motta and Prof. David Karger gave two interesting keynote speeches about what it means to develop Semantic Web applications. They both agreed that it is not enough to consume and/or produce RDF statements to claim that an application is a Semantic Web application; something more is actually needed (e.g. the use of statistics, inference tools and heuristics). In particular, Prof. Karger highlighted a particular requirement that any Semantic Web application must comply with (emphases added by me)[52]:

> A Semantic Web application is *one whose schema is expected to change*. [ . . . ] The fact that the underlying data was RDF might make it easy to change the schema in the data model, but there's no hint of how that change might easily be propagated to the user interface.

In general, this is true for any kinds of Semantic Web application and it is important to take this point into account to develop real and effective Semantic Web applications.

[52] From the Karger's blog post "Keynote at ESWC Part 3: What's Wrong with Semantic Web Research, and Some Ideas to Fix it", available at http://haystack.csail.mit.edu/blog/2013/06/10/keynote-at-eswc-part-3-whats-wrong-with-semantic-web-research-and-some-ideas-to-fix-it/.

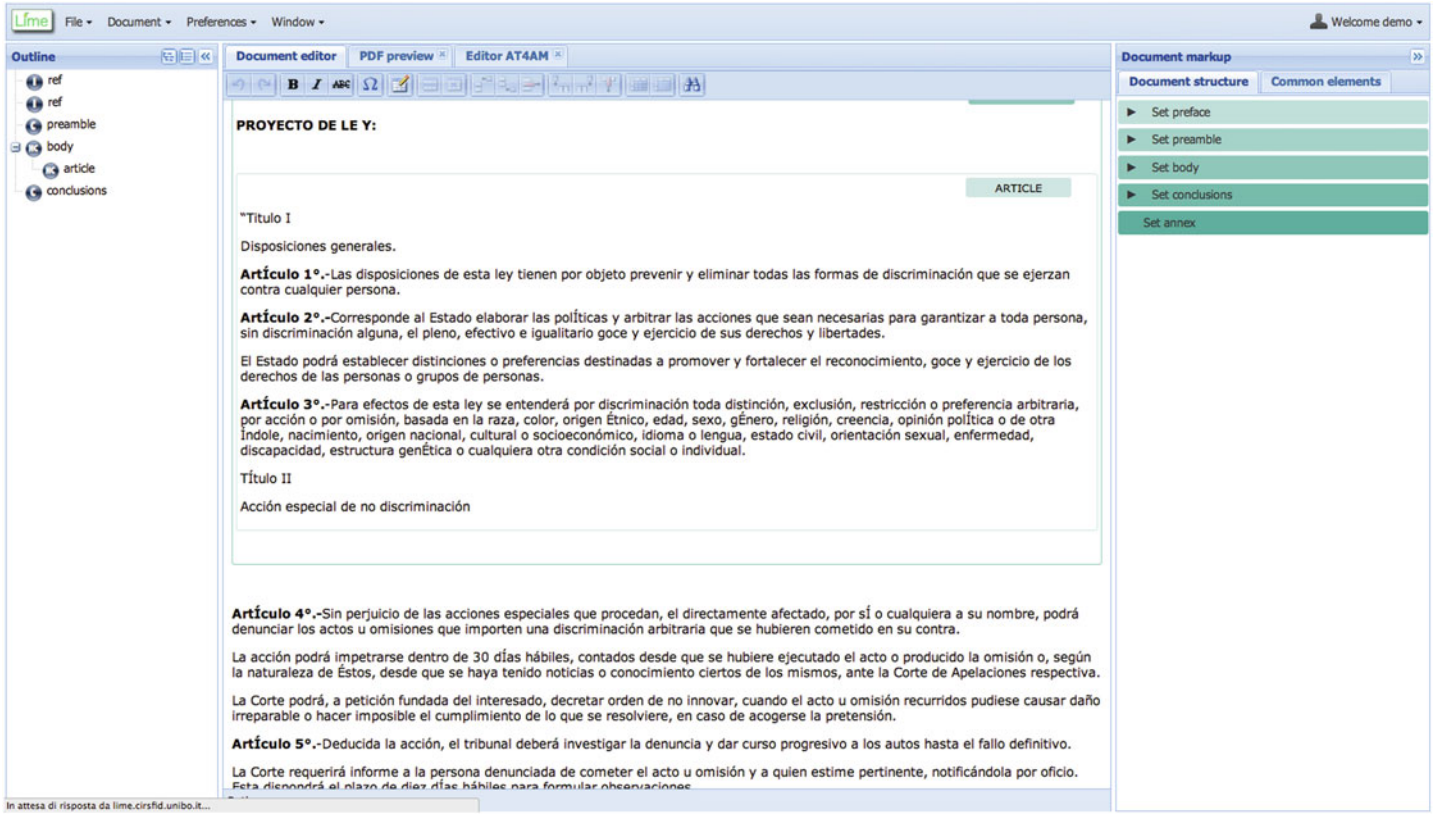


**Fig. 6.23** The LIME application to create Akoma Ntoso markup documents from plain text

Karger's suggestion applies not only to particular Semantic Web languages (RDF, OWL, etc.) but refers also to all those applications that are built upon a specific model that is expected to change in the future.

Along this line, and in the context of applications and editors for legislative documents such as those described in the previous sections, I have identified three specific aspects that must be taken into account in order to develop flexible and long-lasting tools. These I will introduce in the following sections.

#### 6.6.4.1 The Evolution of Legislative Markup Languages

XML is the most commonly used format to store and share textual documents so as to make it easy for machines to access them. Several libraries have been already developed and new tools are built on the basis of this (meta-)language. The choice to develop Akoma Ntoso as an XML-based language followed directly from these premises.

As introduced in Sects. 4.1 and 2.2.6, Akoma Ntoso is an OASIS international standard that is broadly adopted as main format to store legislative documents by different institutions and organisms, such as the European Commission; the European Parliament; the Italian Senate, and several African Parliaments, and the adoption process is set to continue.

Of course, different versions of Akoma Ntoso have been released in the past and more and will be released in the next few years to address issues and/or features deriving from particular (even new) scenarios. Since it is a live standard, Akoma Ntoso will continue to evolve during this time. In addition to that, it is also possible that, in the future, other (even new) XML languages will be preferred as standard formats for legislative documents instead of Akoma Ntoso.

The alterations required following a change of the base format for storing documents, which may happen at any point in time in the future, should be handled in an automatic manner by the legislative editors of the time. In particular, such editors should be able, automatically (or semi-automatically), to adapt their interface according to the modifications made in the particular XML language in use, without extensive effort by the developers and, in particular, the users of such editors.

The developers of LIME are currently working on enabling the editor to visualise and process automatically new versions of Akoma Ntoso. However, additional mechanisms should also be considered to enable the automatic adaptation of the user interface when users decide to adopt another markup language instead of Akoma Ntoso.

Of course it is not feasible to propose and develop an automatic approach that takes into account any possible language based on XML. A possible way around this issue could be to require that, whatever markup language is used, it will be strictly compliant with very general rules imposing only a limited but enough expressive structural organisation of document content. In practice, the idea would be to process all those "constrained" markup languages that follow some sort of general and shared design principles only. In that case, the use of such constrained markup languages should make it possible to develop an editor that adapts his interface to the

new language in consideration without being concerned by its vocabulary and the particular content model of its markup elements.

A possible theory, which can be applied to XML-based languages to restrict their expressiveness to practical and circumscribed design principles, is the one about structural patterns, that I introduced in Sect. 3.3.2. As asserted that Section, a small number of patterns is sufficient to express what most users need to allow one to capture the most relevant document structures. Thus, in this case, it will be possible, in principle, to develop an editor that manages markup according to only eleven possible kinds of structures—which have precise structural and presentational semantics that act independently from the name and the content model of markup elements, as defined in the markup language in consideration. The development of an editor according to this theory will enable an easy interchange between the currently-used markup language and other languages, without being concerned about their implementation or the re-writing of pieces of programming code to adapt the editor application to a new markup format.

#### 6.6.4.2 Use your Favourite Ontologies

The interlinking between legal and legislative content and external entities, such as those in the Linked Open Data, is an important feature that legislative editors should guarantee. As shown in Sect. 4.4, the use of external entities and ontologies to enrich document content can be very useful in real life scenarios, such as legal drafting, reference browsing, document analysis, and similar.

However, while the choice of a particular markup language for legislative documents is partially constrained by the few recognised standards already used world-wide, there are no standardised Semantic Web ontologies that are more commonly by official institutions. In addition, the ontologies developed in the past usually handle a particular subdomain of the whole legal and legislative domain, as introduced in Sect. 2.4, thus making often simultaneous use of several ontologies in order to model a legal/legislative scenario fully. For instance, in Sect. 4.4 I used ALLOT (see Sect. 2.4.4) and DBPedia (Auer et al. 2007) to model certain concepts, whereas I could have used LKIF (see Sect. 2.4.3) instead of ALLOT or both these ontologies plus DBPedia, and so on.

The choice of what ontologies to use in legislative editors, in order to open up to the semantic enrichment of legislative text, relates strictly to the users' subjective interpretations of such a text. Thus, the editor may suggest but should not impose the use of specific ontologies and particular interfaces to semantically-enhance legislative documents. Rather, it should make possible to change the ontologies and the related interface used quickly and easily to enable the users the annotation of such legislative documents with ontology entities.

The use of existing approaches and principles, such as Gaffe's (discussed in Sect. 6.5), may be very helpful in this direction by allowing one to reduce drastically the time and the complexity needed to reconure the legislative editor according to evolving users' needs.

#### 6.6.4.3 Multilingualism as a Key Factor for a Broad Adoption

The intended audience of such legislative editors is actually very broad. They are used by people having different skills and are adopted by different countries, each with their own rules, laws and, in particular, their official language.

The multilingualism of tools and models is a very important and crucial aspect to consider when targeting a broad audience. Both Web and Semantic Web disciplines have been discussing how to use simultaneously entities (such as HTML pages and RDF resources) written in different languages, so as to facilitate their interaction with multilingual technologies and localisation processes. The interest of the academic community in this topic is confirmed by a number of recent initiatives, e.g. the W3C *MultilingualWeb-LT Working Group*[53] (which recently released a Working Draft of the *Internationalization Tag Set* (version 2; McCane et al. 2013)), the *Multilingual Semantic Web Workshop* series (Buitelaar et al. 2010, 2012; Montiel-Ponsoda et al. 2011) and the *Multilingual Web Initiative*[54].

Starting from these premises, a legislative editor should address the problem of multilingualism from different perspectives. First of all, its interface should be dynamically and automatically adaptable to the particular native language of users, and have all the menus, icons, manuals, markup elements, ontological entities, and the like, shown in the target language rather than only in English. Secondly, it should be guaranteed the interoperability between the legislative markup format in use and other formats, especially when the latter are expressed in different languages – for instance, Akoma Ntoso (see Sect. 4.1) in English and Norme in Rete (see Sect. 2.2.2) in Italian. Finally, the same degree of interoperability should be also guaranteed between the adoption of ontologies, developed in different languages according to specific purposes.

Addressing these aspects and all those introduced in the previous sections will enable legislative editors to be the flexible and broadly-adopted Semantic Web-aware applications of the future in the context of the legal domain.

## References

Auer, S., S. Dietzold, and T. Riechert. 2006. OntOWiki—A tool for social, semantic collaboration. In Proceedings of the 5th international semantic web conference (ISWC 2006), eds. I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, lecture notes in computer science, vol. 4273, 736–749. Berlin: Springer. doi:10.1007/11926078_53.

Auer, S., C. Bizer, J. Lehmann, G. Kobilarov, R. Cyganiak, and Z. Ives. 2007. DBpedia: A nucleus for a web of open data. In Proceedings of 6th international semantic web conference and of the 2nd Asian semantic web conference (ISWC 2007 + ASWC 2007), eds. K. Aberer, K. Choi, N. F. Noy, D. Allemang, K. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi,

[53] MultilingualWeb-LT Working Group homepage: http://www.w3.org/International/multilingualweb/lt/.

[54] Multilingual Web Initiative homepage: http://www.multilingualweb.eu.

G. Schreiber, and P. Cudré-Mauroux, lecture notes in computer science, vol. 4825, 722–735. Berlin: Springer. doi:10.1007/978-3-540-76298-0_52.

Bao, J., P. R. Smart, N. Shadbolt, D. Braines, and G. Jones. 2009. A controlled natural language interface for semantic media wiki. In Proceedings of the 3rd annual conference of the International Technology Alliance (ACITA2009), 23–24 Sept 2009. Maryland, USA. http://www.usukita.org/papers/4551/SemanticWikiv7.pdf. Accessed 30 July 2013.

Basca, C., S. Corlosquet, R. Cyganiak, S. Fernández, and T. Schandl. 2008. Neologism: easy vocabulary publishing. In Proceedings of the 4th workshop on scripting for the semantic web, eds. C. Bizer, S. Auer, G. A. Grimnes, and T. Heath. CEUR workshop proceedings, vol. 368. Aachen: CEUR-WS-org. http://ceur-ws.org/Vol-368/paper10.pdf. Accessed 30 July 2013.

Beckett, D. 2004. RDF/XML syntax specification (revised). W3C recommendation, 10 Feb 2004. World Wide Web Consortium. http://www.w3.org/TR/rdf-syntax-grammar/. Accessed 30 July 2013.

Berrueta, D., and J. Phipps. 2008. Best practice recipes for publishing RDF vocabularies. W3C working group note 28 Aug 2008. World Wide Web Consortium. http://www.w3.org/TR/swbp-vocab-pub/. Accessed 30 July 2013.

Birks, M., and J. Mills. 2011. *Grounded theory: A practical guide*. Thousand Oaks: SAGE. (ISBN: 978-1848609938).

Bojars, U., and J. G. Breslin. 2010. SIOC core ontology specification. 25 March 2010. http://rdfs.org/sioc/spec/. Accessed 30 July 2013.

Boley, H., G. Hallmark, M. Kifer, A. Paschke, A. Polleres, and D. Reynolds. 2013. RIF core dialect. 2nd ed.. W3C recommendation, 5 Feb 2013. World Wide Web Consortium. http://www.w3.org/TR/rif-core/. Accessed 30 July 2013.

Bolognini, V., A. Di Iorio, S. Duca, A. Musetti, S. Peroni, and F. Vitali. 2009. Exploiting ontologies to deploy user-friendly and customized metadata editors. In Proceedings of the IADIS Internet/WWW 2009 conference, eds. B. White, P. Isaías, and M. B. Nunes. Lisbon: IADIS.

Brockmans, S., R. Volz, A. Eberhart, and P. Löffler. 2004. Visual modeling of OWL DL ontologies using UML. In Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), eds. S. A. McIlraith, D. Plexousakis, and F. van Harmelen, lecture notes in computer science, vol. 3298, 198–213. Berlin: Springer. doi:10.1007/978-3-540-30475-3_15.

Brockmans, S., P. Haase, P. Hitzler, and R. Studer. 2006. A metamodel and UML profile for rule-extended OWL DL ontologies. In Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), eds. Y. Sure and J. Domingue, lecture notes in computer science, vol. 4011, 303–316. Berlin: Springer. doi:10.1007/11762256_24.

Brooke, J. 1996. SUS: A "quick and dirty" usability scale. In *Usability evaluation in industry*, eds. P. W. Jordan, B. Thomas, B. A. Weerdmeester, and A. L. McClelland, 189–194. London: Taylor and Francis. (ISBN: 0748404600).

Buffa, M., F. Gandon, G. Ereteo, P. Sander, and C. Faron. 2008. SweetWiki: A semantic wiki. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 6 (1): 84–97. doi:10.1016/j.websem.2007.11.003.

Buitelaar, P., P. Cimiano, and E. Montiel-Ponsoda, eds. 2010. Proceedings of the 1st international workshop on the Multilingual Semantic Web (MSW 2010). CEUR workshop proceedings, vol. 571. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-571/. Accessed 30 July 2013.

Buitelaar, P., P. Cimiano, D. Lewis, J. Pustejovsky, and F. Sasaki, eds. 2012. Proceedings of the 3rd international workshop on the Multilingual Semantic Web (MSW 2012). CEUR workshop proceedings 936. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-936/. Accessed 30 July 2013.

Carroll, J., I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. 2004. Jena: Implementing the semantic web recommendations. In Proceedings of the 13th international conference on World Wide Web—Alternate track papers & posters (WWW 2004), eds. S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, 74–83. New York: ACM. doi:10.1145/1013367.1013381.

d'Aquin, M., and E. Motta. 2011. Watson, more than a semantic web search engine. *Semantic Web—Interoperability, Usability, Applicability* 2 (1): 55–63. doi:10.3233/SW-2011-0031.

De Coi, J. L., N. E. Fuchs, K. Kaljurand, and T. Kuhn. 2009. Controlled English for reasoning on the semantic web. In *Semantic techniques for the web—The REWERSE perspective*, eds. F. Bry and J. Maluszynski, 276–308. (ISBN: 3642045806).

Dello, K., E. B. S. Paslaru, and R. Tolksdorf. 2006. Creating and using semantic web information with makna. In Proceedings of the 1st workshop on semantic wikis—From wiki to semantics, eds. M. Völkel and S. Schaffert. CEUR workshop proceedings, vol. 206. Aachen: CEUR-WS.org. http://www.ceur-ws.org/Vol-206/paper4.pdf. Accessed 30 July 2013.

Di Iorio, A., A. Musetti, S. Peroni, and F. Vitali. 2010a. Crowdsourcing semantic content: A model and two applications. In Proceedings of the 3rd international conference on human system interaction (HSI10), ed. T. Pardela, 563–570. Washington DC: IEEE Computer Society. doi:10.1109/HSI.2010.5514513.

Di Iorio, A., A. Musetti, S. Peroni, and F. Vitali. 2010b. Ontology-driven generation of wiki content and interfaces. *New Review of Hypermedia and Multimedia* 16 (1): 9–31. doi:10.1080/13614568.2010.497194.

Di Iorio, A., A. Musetti, S. Peroni, F. Vitali. 2011a. OWiki: Enabling an ontology-led creation of semantic data. In *Human-computer systems interaction: Backgrounds and applications 2,* eds. Z. S. Hippe, J. L. Kulikowski, and T. Mroczek. Berlin: Springer. (ISBN: 3642231711).

Di Iorio, A., S. Peroni, and F. Vitali. 2011b. A semantic web approach to everyday overlapping markup. *Journal of the American Society for Information Science and Technology* 62 (9): 1696–1716. doi:10.1002/asi.21591.

Dragan, L., S. Handschuh, and S. Decker. 2011. The semantic desktop at work: Interlinking notes. In Proceedings the 7th international conference on semantic systems (I-SEMANTICS 2011), eds. C. Ghidini, A. Ngonga Ngomo, S. N. Lindstaedt, and T.Pellegrini, 17–24. New York: ACM. doi:10.1145/2063518.2063521.

Dublin Core Metadata Initiative. 2012a. DCMI metadata terms. DCMI recommendation. http://dublincore.org/documents/dcmi-terms/. Accessed 30 July 2013.

Dublin Core Metadata Initiative. 2012b. Dublin core metadata element set, version 1.1. DCMI recommendation. http://dublincore.org/documents/dces/. Accessed 30 July 2013.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1994. *Design patterns: Elements of reusable object-oriented software*. Boston: Addison-Wesley. (ISBN: 0201633610).

Gardenfors, P. 2000. *Conceptual spaces: The geometry of thought*. Cambridge: Bradford Books. (ISBN: 0262071991).

Garlik, S. H., and A. Seaborne. 2013. SPARQL 1.1 Query language. W3C recommendation 21 March 2013. World Wide Web Consortium. http://www.w3.org/TR/sparql11-query/. Accessed 30 July 2013.

Gasevic, D., D. Djuric, V. Devedzic, and V. Damjanovic. 2004. Converting UML to OWL ontologies. In Proceedings of the 13th international World Wide Web conference on alternate track papers & posters (WWW Alt 04), 488–489. New York: ACM. doi:10.1145/1013367.1013539.

Harth, C., H. Gassert, I. O'Murchu, J. G. Breslin, and S. Decker. 2005. WikiOnt: An ontology for describing and exchanging Wikipedia articles. In Proceedings of Wikimania 2005, eds. J. Voss, A. Lih, S. Klein, and C. Ma. http://meta.wikimedia.org/wiki/Wikimania05/Paper-IM1. Accessed 30 July 2013.

Horridge, M., and S. Bechhofer. 2011. The OWL API: A Java API for OWL ontologies. *Semantic Web—Interoperability, Usability, Applicability* 2 (1): 11–21. doi:10.3233/SW-2011-0025.

Horridge, M., and P. Patel-Schneider. 2012. OWL 2 Web ontology language Manchester syntax. 2nd ed. W3C working group note 11 Dec 2012. World Wide Web Consortium. http://www.w3.org/TR/owl2-manchester-syntax/. Accessed 30 July 2013.

Katifori, A., E. Torou, C. Halatsis, G. Lepouras, and C. Vassilakis. 2006. A comparative study of four ontology visualization techniques in Protege: Experiment setup and preliminary results. In Proceedings of the 10th international conference on information visualisation (IV 2006), eds. E. Banissi, R. A. Burkhard, A. Ursyn, J. J. Zhang, M. Bannatyne, C. Maple, A. J. Cowell, G. Y. Tian, and M. Hou, 417–423. Washington DC: IEEE Computer Society. doi:10.1109/IV.2006.3.

Katifori, A., C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. 2007. Ontology visualization methods—A survey. ACM Computing Surveys, 39 (4): Article 10. doi:10.1145/1287620.1287621.

Kay, J., and A. Lum. 2003. An ontologically enhanced metadata editor. University of Sidney, School of Information Technologies, Technical Report 541. http://www.it.usyd.edu.au/research/tr/tr541.pdf. Accessed 30 July 2013.

Klein, B., C. Höcht, and B. Decker. 2005. Beyond capturing and maintaining software engineering knowledge—"Wikitology" as shared semantics. Presented during the workshop on Knowledge Engineering and Software Engineering (KESE 2005). 11 Sept 2005, Koblenz, Germany. http://www.dfki.uni-kl.de/klein/papers/finalKESE05.pdf. Accessed 30 July 2013.

Knublauch, H., M. Horridge, M. A. Musen, A. L. Rector, R. Stevens, N. Drummond, P. W. Lord, N. F. Noy, J. Seidenberg, and H. Wang. 2005. The Protege OWL experience. In Proceedings of the OWLED 05 Workshop on OWL: Experiences and directions, eds. B. C. Grau, I. Horrocks, B. Parsia, P. F. Patel-Schneider. CEUR workshop proceedings, vol. 188. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-188/sub14.pdf. Accessed 30 July 2013.

Kriglstein, S., and R. Motschnig-Pitrik. 2008. Knoocks: New visualization approach for ontologies. In Proceedings of the 12th international conference on Information Visualisation (IV 2008), 163–168. Washington DC: IEEE Computer Society. doi:10.1109/IV.2008.16.

Lakoff, G. 1987. *Women, fire, and dangerous things*. Chicago: University Of Chicago Press. (ISBN: 0226468046).

Lakoff, G., and M. Johnson. 1980. *Metaphors we live by*. Chicago: University Of Chicago Press. (ISBN: 0226468011).

Lewis, J. R., and J. Sauro. 2009. The factor structure of the system usability scale. In Proceedings of the 1st international conference on human centered design (HCD 09), ed. M. Kurosu, lecture notes in computer science, vol. 5619, 94–103. Berlin: Springer. doi:10.1007/978-3-642-02806-9_12.

Li, N., E. Motta, and M. d'Aquin. 2010. Ontology summarization: An analysis and an evaluation. In Proceedings of the International Workshop on Evaluation of Semantic Technologies (IWEST 2010), eds. A. Gomez-Perez, F. Ciravegna, F. van Harmelen, and J. Hefflin. CEUR workshop proceedings, vol. 666. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-666/paper8.pdf. Accessed 30 July 2013.

McCane, S., D. Lewis, C. Lieske, A. Lommel, J. Kosek, F. Sasaki, and Y. Savourel. 2013. Internationalization tag set (ITS) version 2.0. W3C recommendation 29 Oct 2013. http://www.w3.org/TR/its20/. Accessed 30 July 2013.

Montiel-Ponsoda, E., J. McCrae, P. Buitelaar, P. Cimiano, eds. 2011. Proceedings of the 2nd international workshop on the Multilingual Semantic Web (MSW 2011). CEUR workshop proceedings, vol. 775. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-775/. Accessed 30 July 2013.

Motta, E., S. Peroni, N. Li, and M. D'Aquin. 2010. KC-Viz: A novel approach to visualizing and navigating ontologies. In Proceedings of the EKAW2010 poster and demo track, eds. J. Völker and O. Corcho. CEUR workshop proceedings, vol. 674. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-674/Paper191.pdf. Accessed 30 July 2013.

Motta, E., P. Mulholland, S. Peroni, M. D'Aquin, J. M. Gomez-Perez, V. Mendez, and F. Zablith. 2011a. A novel approach to visualizing and navigating ontologies. In Proceedings of the 10th international semantic web conference (ISWC 2011), eds. L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, lecture notes in computer science, vol. 7031, 470–486. Berlin: Springer. doi:10.1007/978-3-642-25073-6_30.

Motta, E., S. Peroni, and M. d'Aquin. 2011b. Latest developments in KC-Viz. In Proceedings of the poster & demo track of the 10th International Semantic Web Conference (ISWC 2011). http://iswc2011.semanticweb.org/fileadmin/iswc/Papers/PostersDemos/iswc11pd_submission_22.pdf. Accessed 30 July 2013.

Motta, E., S. Peroni, J. M. Gómez-Pérez, M. d'Aquin, and N. Ning Li. 2012. Visualizing and navigating ontologies with KC-Viz. *Ontology engineering in a networked world*, eds. M.

C. Suárez-ueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, 343–362. Berlin: Springer. doi:10.1007/978-3-642-24794-1_16.

Object Management Group. 2009. Ontology Definition Metamodel (ODM) version 1.0. http://www.omg.org/spec/ODM/1.0/PDF. Accessed 30 July 2013.

Object Management Group. 2011. Unified modeling language™ (UML®). http://www.omg.org/spec/UML/2.4/. Accessed 30 July 2013.

Oren, E., R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. 2008. Sindice.com: A document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies* 3 (1): 37–52. doi:10.1504/IJMSO.2008.021204.

Palmirani, M., L. Cervone, O. Bujor, and M. Chiappetta. 2013. RAWE: An editor for rule markup of legal texts. In joint Proceedings of the 7th international rule challenge, the special track on human language technology and the 3rd RuleML doctoral consortium. CEUR workshop proceedings, vol. 1004. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-1004/paper4.pdf. Accessed 30 July 2013.

Passant, A., and P. Laublet. 2008. Towards an interlinked semantic wiki farm. In Proceedings of the 3rd semantic wiki workshop (SemWiki 2008), eds. C. Lange, S. Schaffert, H. Skaf-Molli, and M. Völkel. CEUR workshop proceedings, vol. 360, 1–14. Aachen: CEUR-WS.org. http://ceur-ws.org/Vol-360/paper-19.pdf. Accessed 30 July 2013.

Peroni, S., E. Motta, and M. d'Aquin. 2008. Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures. In Proceedings of the 3rd Asian semantic web conference (ASWC 2008), eds. J. Domingue and C. Anutariya, lecture notes in computer science, vol. 5367, 242–256. Berlin: Springer. doi:10.1007/978-3-540-89704-0_17

Peroni, S., D. Shotton, and F. Vitali. 2012. The Live OWL documentation environment: A tool for the automatic generation of ontology documentation. In Proceedings of the 18th international conference on knowledge engineering and knowledge management (EKAW 2012), eds. A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Aquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, lecture notes in computer science, vol. 7603, 398–412. Berlin: Springer. doi:10.1007/978-3-642-33876-2_35.

Peroni, S., D. Shotton, and F. Vitali. 2013. Tools for the automatic generation of ontology documentation: A task-based evaluation. *International Journal on Semantic Web and Information Systems* 9 (1): 21–44. doi:10.4018/jswis.2013010102.

Plaisant, C., J. Grosjean, and B. B. Bederson. 2002. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In Proceedings of the IEEE symposium on Information Visualization (InfoVis 2002), 57–64. Washington DC: IEEE Computer Society. doi:10.1109/INFVIS.2002.1173148.

Rosch, E. 1978. Principles of categorization. In *Cognition and categorisation,* eds. E. Rosch and B. Lloyd. Hillsdale: Lawrence Erlbaum. (ISBN: 0470263778).

Sauro, J. 2011. A practical guide to the system usability scale: Background, benchmarks & best practices. (ISBN: 978-1461062707).

Schaffert, S. 2006. IkeWiki: A semantic wiki for collaborative knowledge management. In Proceedings of 15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2006), 388–396. Washington DC: IEEE Computer Society. doi:10.1109/WETICE.2006.46.

Shneiderman, B. 1992. Tree visualization with tree-Maps: A 2-d space-filling approach. *ACM Transactions on Graphics* 11 (1): 92–99. doi:10.1145/102377.115768.

Shneiderman, B. 1996. The eyes have it: A task by data type taxonomy for information visualizations. In Proceedings of the 1996 IEEE symposium on visual languages (VL 96), 336–343. Washington DC: IEEE Computer Society. doi:10.1109/VL.1996.545307.

Sirin, E., B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. 2007. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 5 (2): 51–53. doi:10.1016/j.websem.2007.03.004.

Souza, K., A. Dos Santos, and S. Evangelista. 2003. Visualization of ontologies through hypertrees. In Proceedings of the Latin American conference on human-computer interaction (CLIHC 03), 251–255. New York: ACM.

Souzis, A. 2005. Building a semantic wiki. *IEEE Intelligent Systems* 20 (5): 87–91. doi:10.1109/MIS.2005.83.

Sowa, J. F. 1987. Semantic networks. In *Encyclopedia of artificial intelligence,* ed. S. C. Shapiro. New York: Wiley. (ISBN: 0471503053).

Storey, M. A., M. A. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. F. Noy. 2001. Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protege. Presented during the K-CAP 2001 workshop on interactive tools for knowledge capture, 20 Oct 2001, Victoria, Canada. http://www.isi.edu/b̃lythe/kcap-interaction/papers/storey.pdf. Accessed 30 July 2013.

Strauss, A., and J. Corbin. 1998. *Basics of qualitative research techniques and procedures for developing grounded theory*. 2nd ed. London: Sage. (ISBN: 978-0803959408).

Suárez-ueroa, M. C., A. Gómez-Pérez, E. Motta, and A. Gangemi. 2012. *Ontology engineering in a networked world*. Berlin: Springer. ISBN: 3642247934.

Tartir, S., I. B. Arpinar, M. Moore, A. P. Sheth, and B. Aleman-Meza. 2006. OntoQA: Metric-based ontology quality analysis. Presented at the workshop on knowledge acquisition from distributed, autonomous, semantically heterogeneous data and knowledge sources, 27 Nov 2005, Houston, Texas, USA. http://lsdis.cs.uga.edu/library/download/OntoQA.pdf. Accessed 30 July 2013.

Tejo-Alonso, C., D. Berrueta, L. Polo, and S. Fernandez. 2011. Metadata for web ontologies and rules: Current practices and perspectives. In Proceeding of the 5th interational conference on Metadata and Semantic Research (MTSR 2011), eds. E. García-Barriocanal, Z. Cebeci, M. C. Okur, and A. Öztürk. Berlin: Springer.

Vrandecic, D., and M. Krötzsch. 2006. Reusing ontological background knowledge in semantic wikis. In Proceedings of the 1st semantic wiki workshop—From wiki to semantics (SemWiki 2008), eds. M. Völkel and S. Schaffert. CEUR workshop proceedings, vol. 206. Aachen: CEUR-WS.org. http://www.ceur-ws.org/Vol-206/paper2.pdf. Accessed 30 July 2013.

Všlkel, M., M. Krštzsch, D. Vrandecic, H. Haller, and R. Studer. 2006. Semantic wikipedia. In Proceedings of the 15th international conference on World Wide Web (WWW 2006), eds. L. Carr, D. De Roure, A. Iyengar, C. A. Goble, and M. Dahlin, 585–594. New York: ACM. doi:10.1145/1135777.1135863.

Wang, T. D., and B. Parsia. 2006. Cropcircles: Topology sensitive visualization of Owl class hierarchies. In Proceedings of the 5th International Semantic Web Conference (ISWC 2006), eds. I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, lecture notes in computer science, vol. 4273, 695–708. Berlin: Springer. doi:10.1007/11926078_50.

Woods, W. A. 1975. What's in a link: Foundations for semantic networks. In *Representation and understanding: Studies in cognitive science,* eds. D. Bobrow and A. Collins. New York: Academic. (ISBN: 0121085503).