

pOWL – A Web Based Platform for Collaborative Semantic Web Development

Sören Auer

University of Leipzig
auer@informatik.uni-leipzig.de

Abstract: With the Ontology Web Language (OWL), a semantic web language exists for knowledge representation on the Web. Although there are some application programming interfaces and tools available for OWL, a framework for parsing, storing, querying, manipulating, serving and serializing OWL knowledge bases in a collaborative web enabled environment for the most distributed web application platform (LAMP – Linux / Apache / MySQL / PHP) was still missing. This paper gives an overview of pOWL, a web base collaborative semantic web development platform for PHP, which has the mentioned characteristics. As a use case, we present the application of pOWL to semantic web content management and how it may be used as a foundation framework for semantic web applications.

1 Introduction

The broad application of ontologies as shared terminological knowledge representations is one of the main strategies of the semantic web paradigm. With OWL (Web Ontology Language, [2]) there exists now a W3C standard for defining web enabled ontologies which fits in the semantic layering of web languages [4].

Although there are some OWL-editors available, most of them are complicated to deploy or handle, do not support strategies for collaborative, distributed development of ontologies, are not Open Source or are not available for the most distributed web technologies. Since PHP [15] is the most distributed web development technology, the semantic web paradigm will probably only be successful in a broad perspective if there are applications and tools available tightly interacting with this language. The goal of this paper is to present a software framework named pOWL [1] which meets this requirement. The aim of pOWL is to deliver an easy-to-deploy and easy-to-use, scalable, PHP and web-based ontology management solution to the Open Source community, which covers the whole ontology lifecycle. Despite the fact that pOWL is still in beta quality stage it is already productively used in several projects.

The paper is organized as follows: After introducing the overall architecture of the pOWL framework in the first section, crucial pOWL features, such as model evolution and versioning, customizability by software knowledge, the user interface and some data concerning scalability, are presented. In addition an approach how the pOWL framework then may be used to build semantic web applications is exemplarily demonstrated by the use case of a semantic web content management solution.

2 Architecture

pOWLs architecture consists of 6 stacked tiers, while trying to minimize dependencies and supplying clean interfaces between tiers. It consists of the following tiers:

- *pOWL store* – SQL compatible relational database backend
- *RDFAPI, RDFSAPI, OWLAPI* – layered APIs for handling RDF, RDF-Schema (RDFS) and OWL
- *pOWL API* – containing classes and functions to build web applications on top of those APIs
- *User interface* – a set of PHP pages combining widgets provided by pOWL API for accessing (browsing, viewing, editing) model data in a pOWL store.

The generic API components are described in more detail in the remainder of this section, while more pOWL specific functionality and the user interface are presented in Sections 3 to 7.

pOWL Store

Any SQL compatible relational database supported by AdoDB [9], which is the database abstraction layer used by pOWL, may act as a pOWL store. The following database tables are used to store all information related to ontologies and their evolution:

Table	Description
models	provides information about the models in the store
statements	contains all statements of models in the store
log_actions	holds information about editing actions on a model
log_statements	contains added and removed statements for every action

The pOWL store uses a denormalized database schema, where all resources and literals are written in full in a table row representing an RDF statement. Tests done by the RDFAPI developers state in [12] that this is 2 to 3 times faster than a normalized database schema where resources and literal values are stored separately. The pOWL store is accessed by RDFAPI.

RDFAPI

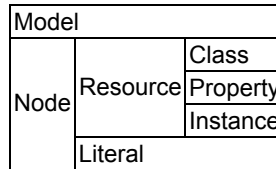
RAP – RDFAPI for PHP [12] – is an independent project by Chris Bizer, Radoslaw Oldakowski and others. It provides the following functionality to pOWL:

- Parser, serializer for different RDF serializations (XML, N3, N-Triple)
- RDQL declarative query backend
- Classes and methods for working with RDF models, resources and literals
- NetAPI for publishing models on the web.

The higher layered APIs, RDFSAPI and OWLAPI, extend the classes “Model”, “Resource” and “Literal” provided by RDFAPI.

RDFSAPI

RDFSAPI extends RDFAPI’s class schema by RDF-Schema [8] specific classes as shown in the following figure:



RDFSAPI tries to apply the interpreted languages approach of “typelessness”. Whenever a resource (class, instance, property respectively) is requested (e.g. as a function parameter) the following options representing the resource are available:

- RDFSResource object
- Local name (as a string, e.g. “Article”)
- URI (as a string, e.g. “http://purl.org/net/nknouf/ns/bibtex#Article”)
- Namespace prefix and local name (as a string, e.g. “bibtex:Article”)

The following table gives an overview over methods exposed by the main RDFSAPI classes. Each value in curly brackets stands in conjunction with the prefix (as “add” or “list”) for a different method. A complete API documentation including required parameters for each method and detailed explanations can be found at [1].

RDFSModel	RDFSClass
add{Triple,Class,Property,Instance} count{Triples,Classes,Properties,Instances} find{Triples,Resources} get{Triple,Resource,Class,Property,Instance} listTop{Classes,Properties} list{Resources,Classes,Properties,Instances} list{Namespaces,Languages,Datatypes} remove{Triple,Resource,Class,Property,Instance} load	add{Instance,Property} count{Instances,Subclasses} find{Instances} get{Instance} list{SubClasses,SuperClasses,Properties,Instances} remove{Instance,Property} set{SubClasses,SuperClasses,Properties}
RDFSProperty	RDFSInstance
add{Domain,Range} add{SubProperty,SuperProperty} get{Domain,Range} list{Domain,Range} list{SubProperties,SuperProperties} remove{Domain,Range} set{Domain,Range}	addPropertyValue get{Class,Classes} get{PropertyValue} listProperties list{PropertyValues} setProperty{Value,Values}

OWLAPI

All classes of RDFSAPI are extended with methods for handling OWL predefined properties, DL axioms and restrictions, as well as basic subsumption inference. pOWL doesn't store entailed triples. In case of more abstract ontology languages layered on top of RDFS or OWL this would lead to an explosion of required database storage. Entailment is calculated when needed and cached for reuse.

3 Model Evolution and Versioning

To enable domain experts to collaboratively develop shared conceptualizations based on the Ontology Web Language a key requirement is to support a sophisticated versioning strategy. Every editing action can be decomposed into smaller editing actions and finally into adds and removes of RDF triples to or from the RDF model. The following example illustrates this:

```

Class "owl:wine" updated
  Labels added
    Label for language "de" added: "Wein"
    Statement added:
      <owl:wine> <rdfs:label> "Wein"
    Label for language "ru" added: "Vino"
    Statement added:
      <owl:wine> <rdfs:label> "Vino"
  Annotation property
    "rdfs:seeAlso" changed from "wn-concept:103880346" into
      "http://en.wikipedia.org/wiki/Wine"
    Statement removed:
      <owl:wine> <rdfs:seeAlso> "wn-concept:103880346"
    Statement added:
      <owl:wine> <rdfs:seeAlso>
        "http://en.wikipedia.org/wiki/Wine"

```

pOWL enables rollback of every particular editing action by determining if the involved triples are still present (if added) or still missing (if removed). A parent action thus may only be rolled back if all sub-actions may be rolled back as well.

4 Customizability by Software Knowledge

Instead of using configuration files or special database tables for customization, user authorization and preference management as well as user interface translation purposes and module integration, pOWL uses a “system ontology” for storing such knowledge. This “system ontology” may be edited and managed using pOWL itself. The user or administrator is further generically restricted to apply only valid configurations and this approach expands software flexibility dramatically. Three examples for knowledge in this system model are given:

- Instances of the system ontology class “Label” contain translations for all texts presented on the user interface and keep track of their usage in different parts of the application, thus simplifying the translation.
- Instances of the classes “User” and “Group” are responsible for authorization and storing of preferences.
- Further classes are used to store configuration data of pOWL modules and widget plugins. Widgets for presenting and editing literal data may even be selected and configured dependent on the context they are actually used in (e.g. the user logged in, the data type of the literal or the property which the data value is assigned to).

pOWL may be extended by software modules which are placed in a subfolder of the pOWL distribution. Modules consist of:

- A list of namespaces for which the module should be active – if a resource of the actual model belongs to one of these namespaces the module will be loaded.
- Functions or PHP pages which provide a new view on the knowledge base or expose distinct functionality to be presented in a separate tab.
- Functions or PHP pages which may be called for an arbitrary triple, resource, class, property or instance – these will be linked with the corresponding objects in other modules whenever the user selects a triple, resource, class, property or instance.

The semantic web content management approach presented in Section 8 is implemented as a pOWL module.

5 User Interface and Widgets for Data Editing

The user interface is arranged in tabs, each tab representing a different view on the knowledge base. The following tabs are included in pOWL (additional ones may be added in a modular manner):

- *Models* – provides an overview on the models in pOWLs store.
- *Triples* – displays a browse-able and searchable list of the triples in the selected ontology.
- *Classes* – hierarchically organizes classes and allows viewing and editing their definitions.
- *Properties* – hierarchically organizes properties and allows viewing and editing their definitions.
- *Instances* – gives various views on instances of a distinct class in the model.
- *Version* – provides access to information about the evolution of the ontology

pOWL provides a comprehensive library of plug-ins for comfortable editing of data:

- Single and multiple line text editing
- Radio, checkbox and drop-down widgets

The screenshot displays the pOWL Semantic Web Development Platform interface. The main window shows the 'CabernetFranc' class with its properties and axioms. The properties table is as follows:

Name	Inherited from	Range	Cardinality	Other restrictions	Action
hasColor	Wine	WineColor	functional	Red	[edit]
hasWineDescriptor	Wine	WineTaste, WineColor		[A]	[edit]
madeFromGrape	Wine	WineGrape		[A]	[edit]
hasBody	Wine	WineBody	functional	Medium	[edit]
hasFlavor	Wine	WineFlavor	functional	Moderate	[edit]
hasSugar	Wine	WineSugar	functional	Dry	[edit]
madeFromFood	food ConsumableThing	food SweetFruit, food NonSweetFruit		[A]	[edit]
locatedIn	owl:Thing	Region		Region (Wine)	[edit]

The interface also includes a left sidebar with a class hierarchy, a search bar, and a right sidebar with 'Classes' and 'On the Web' sections. The 'Classes' section provides an overview of the class concept, and 'On the Web' lists related resources like 'Classes (RDF Schema Vocabulary Description Language)', 'Simple Named Classes, Complex Classes (OWL Guide)', 'Classes (OWL Reference)', and 'owl:Class (OWL Semantics)'.

- WYSIWIG HTML editing (HTMLArea component integrated)
- Selection or referencing of arbitrary resources from pOWLs store (may be restricted to classes, instances or properties).

The resource selection widget is equipped with "configurable intelligence": if only a few resources may be potentially selected radio, checkbox or drop-down widgets are used, otherwise browsing and searching is enabled in a separate window. All widgets may be customized by creating instances of a widget style profile class for the widget in the system ontology and connecting this with a suitable OWL property.

RDQL query builder

RDQL is an implementation of an SQL-like query language for RDF. It treats RDF as data and provides query with triple patterns and constraints over a single RDF model [14]. pOWL enables users to freely formulate their queries or use the integrated query builder to question the knowledge base. It constraints the users input to only those values which make up a syntactically correct RDQL query, further more the user may only select resource and namespaces available in the knowledge base.

6 Different Points of View on a Model

From the programmer's as well as from the knowledge engineer's or user's point of view an OWL knowledge base may be seen through quite different glasses:

Triples View

Triples or RDF statements are related to natural language sentences consisting of subject, predicate and object. Subjects, predicates and objects may be resources – universally unique concept or entity identifiers. An object may furthermore consist of data typed literal values – literals. In RDFS and OWL such statements or RDF triples are used to define higher level objects like classes, properties and instances and establish relations between them. pOWL supports this view on the knowledge base by enabling the user to view all RDF statements having a currently selected resource as subject, predicate or object. For the programmer RDFAPI provides an exhaustive set of methods to operate on triples.

Database View

An OWL knowledge base may be seen like an object-relational database. The following table establishes a informal correspondence between concepts in ontological knowledge representation and object-relational databases:

RDFS/OWL concept	(O)RDBMS concept
Classes	Tables
Properties	Cols
Instances	Rows

To view a knowledge base in database manner, a highly configurable and filterable “instance overview” arranges instances of a class in a tabular view. Further instances may be exported and imported in spreadsheet and database compatible CSV format. The programmer is supplied with the SQL inspired declarative query language RDQL and methods for database table row like operations on instances.

Description Logic Axioms View

The following table shows some examples how RDFS/OWL properties correspond to Description Logic concepts:

Sign	RDFS/OWL property	DL concept	Sign
⊆	subClassOf subPropertyOf	Implication	→
	allValuesFrom	For all	∀
∪	unionOf	Disjunction	∨

The pOWL user interface provides special widgets supporting the user in viewing and editing class construction, property restriction and individual identity axioms. These

widgets are bound to corresponding OWLAPI calls providing this functionality on the API level.

Serialization View

For RDF abstract syntax, which is the core of all models in pOWL, there are different serialization formats: RDF/XML, N3, N-Triple. pOWL supports viewing and editing of all parts of the ontology (classes, properties and instances with associated information) in these formats.

7 Scalability

pOWL is designed to work with ontologies of arbitrary size (only limited by disk space). This requires that only those parts of the ontology are loaded into main memory which are required to display the information requested by the user on the screen (to render a web page containing this information). Special efforts are done to realize this for the subsumption tree view of classes. Subtrees are loaded on demand using Javascript and only if not already done. This mechanism even enables rendering virtually infinite trees in a webpage, which may occur if a subclass is declared to be equivalent to one of its super-classes.

The following benchmarks were taken on a recent system (Pentium 4, 2.8 Ghz, 512 MB RAM) with the WAMP (Windows, Apache, PHP, MySQL) software installed. The only optimization of the system was to enable MySQL's query cache. By further optimization (especially OPcode Caching as Zend Accelerator, APC or MMCache provide) significant speed improvements may be possible.

Model	Triple count	Import time	Classes	Subsumption Hierarchy Calculation
Wordnet ¹	473 528	624 s	6	0.30 s
NCI Cancer Ontology ²	463 878	597 s	27 652	0.46 s
UNSPSC ³	82 500	82 s	16 499	1.06 s

8 Use Case – Semantic Web Content Management

To show the flexibility of pOWL's approach a semantic web content management solution is developed prototypically on top of the pOWL framework. A key requirement to modern content management systems is the separation of structure, content

¹ <http://www.semanticweb.org/library/wordnet/>

² <http://www.mindswap.org/2003/CancerOntology/>

³ <http://www.cs.vu.nl/~mcaklein/unspsc/unspsc/>

and style. This is realized by defining different interconnected ontologies for the page structure of the web site, for text and multimedia content and for rendering and style information.

Content

The content ontology contains classes, properties and instances representing the content to be displayed on a web site. As an example we modelled the content structure of an arbitrary university chair with the following interconnected classes:

- *Lecture* – instances describe lectures held at the chair
- *Event* – connects content elements (e.g. a lecture) with concrete dates
- *Publication* – defined to be equivalent to bibtex:Entry⁴
- *Person* – with subclasses for professor, staff, student
- *Address*.

The task of the website editors is to fill these classes with instances. By using the pOWL user interface it is easy for each chair member to maintain his/her own content. Since pOWL generates the appropriate forms to fill the instance property values, it is ensured that no information was forgotten incidentally and relations between content elements remain consistent.

Style

Since all content is stored in property values attached to class instances, it seems natural to relate information, how this content should be presented, to the classes and properties respectively. The style ontology thus connects classes and properties of the content ontology to templates and parameters used to render concrete values attached to these resources. An example template for the class “Address” (with literal valued properties “room”, “street”, “streetNo”, “zip” and “city”) could look as follows:

```
<div class="adress">
  Room: {room}<br />
  Street: {street} {streetNo}<br />
  City: {zip} {city}
</div>
```

If a property does not contain a literal value but references some other class instance, a template for this property is searched to render a view of the target instance.

Structure

The structure ontology now defines a treelike page structure of the website. For this – instances of a class “Page” with properties for “parentPage” and “position” are

⁴ with namespace prefix bibtex bound to <http://purl.org/net/nknouf/ns/bibtex>

created. The resulting page tree is presented to the content manager by the pOWL CMS tab plug-in for easy editing. Further, each page is related (by an object property) to either a class or an instance of the content ontology which should be displayed on this page. If related to a class, a list of instances of this class with links to appropriate details pages will be rendered, otherwise the instance values will be rendered according to the templates defined in the style ontology.

Content Syndication

Since content structure and style are now separated in different models, the content model can be easily accessed by the HTTP protocol using the included RDF NetAPI [11] implementation. Some extra efforts could be undertaken to provide RSS feeds or other content syndication formats.

Versioning

All content management editing may be traced back to editing actions on the underlying ontologies. E.g. if a page is added appropriate class instances in the structure, content and style ontologies are created. By enveloping these editing actions with an additional “Page added” editing action in the same manner as described in section 3 – the content management versioning is a simple extension of the ontology versioning, thus enabling the direct usage of the pOWL versioning tab and related tools in the pOWL framework.

Workflow

In the current prototype workflow functionality is not yet implemented, but it should be easy to attach a property giving information about the workflow status to each class in the content, structure and style ontologies. Of course additional programming efforts have to be undertaken to infer and nicely present tasks to be executed to a distinct editor.

9 Conclusion

For domain experts and knowledge engineers, pOWL provides an easy deployable and easy-to-use, web-based ontology editing and publishing solution. pOWL supports collaborative work with ontologies as well as observing the ontology evolution. pOWL is scalable and can be used even with extremely large knowledge bases. It may be customized according to specific needs.

For PHP developers, pOWL offers a comprehensive framework of functionalities for parsing, storing, querying, manipulating, serving and serializing RDFS and OWL ontologies. As shown in the semantic web content management use case, it may function as a basis for more domain specific web based ontology applications.

pOWL has been downloaded over 300 times since April 2004, when pOWL development started. It is available under GNU Public Licence⁵.

For future releases it is planned to integrate more powerful inferencing capabilities and to apply pOWL for OWL-S [13] and web service development.

Acknowledgements

I would like to thank the pOWL development team members, especially Norman Beck, for contributing to pOWL development and Hesham Khalil, Patry Burek and Thies Jochimsen for many inspiring discussions and their support considering this paper.

References

1. Auer, S.: pOWL Homepage, The Web, 2004, <http://powl.sourceforge.net/>
2. Bechhofer, S., Dean, M., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L.: Web Ontology Language (OWL) Reference version 1.0. Recommendation, W3C (2004), <http://www.w3.org/TR/owl-ref/>.
3. Beckett, D.: RDF/XML Syntax Specification (Revised). W3C. 10 February 2004.
4. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 284 (2001) 35–43
5. Carroll, J., et. al: Jena: Implementing the Semantic Web Recommendations. Bristol. 2003. <http://www.hpl.hp.com/techreports/2003/HPL-2003-146.pdf>
6. Donini, F.M., Lenzerini, M., Nardi, D., , Schaerf, A.: Reasoning in description logics. In Brewka, G., ed.: Principles of Knowledge Representation. Studies in Logic, Language and Information. CSLI Publications, (1996) 193–238
7. Klyne, G., Carroll, J.: Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C, 10 February 2004.
8. Lassila, O., Swick, R.R.: Resource description framework (RDF) Model and syntax specification. Recommendation, W3C, February 1999. <http://www.w3.org/TR/1999/RECRdf-syntax-19990222>
9. Lim, J.: ADOdb Library for PHP, <http://php.weblogs.com/ADODB>
10. McBride, B: "Jena: Implementing the RDF Model and Syntax Specification", in: Steffen Staab et al (eds.): "Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001", May 2001
11. Moore, G., Seaborne, A.: RDF Net API, W3C Member Submission, 2. October 2003. <http://www.w3.org/Submission/2003/SUBM-rdf-netapi-20031002/>
12. Oldakowski, R., Bizer, Chr.: RAP: RDF API for PHP, To be published in Proceedings of the "1st International Workshop on Interpreted Languages", 2004.
13. OWL Services Coalition: OWL-S: Semantic Markup for Web Services, The Web, 2004, <http://www.daml.org/services/owl-s/1.0/>
14. Seaborne, A.: RDQL - A Query Language for RDF, W3C Member Submission, 9 January 2004. <http://www.w3.org/Submission/RDQL/>
15. The PHP Group: PHP Homepage, <http://www.php.net>.

⁵ <http://www.gnu.org/copyleft/gpl.html>