Proposal of a Prolog-based Knowledge Wiki[⋆]

Grzegorz J. Nalepa and Igor Wojnicki

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
`gjn@agh.edu.pl`, `wojnicki@agh.edu.pl`

Abstract This paper presents a proposal of a knowledge wiki system and describes its prototype implementation. The system design is based on a wiki concept regarding authoring policies and user access, extended with an ability of storing knowledge in addition to the human-readable content. The knowledge is expressed in the Prolog language by means of predicate logic. An inference engine is coupled with the wiki providing means for an automated knowledge processing and interpretation. Selected applications, examples, and further extensions are also presented.

1 Introduction

The Internet has become a single most important resource for instant information sharing. From the point of view of ordinary users it can be considered to constitute a very flexible and powerful version of the so-called blackboard architecture. The rapid growth of the Internet prompted a rapid development of specific software. This software addresses different aspects of information organization and interchange related issues, such as storing, retrieval, searching, indexing, aggregating, sharing, updating, etc. Taking into account the size and flexibility of the system as a whole, these problems constitute a real challenge.

Today web technologies constitute an advanced and universal programming framework. This framework has a very heterogeneous structure including: data encoding and structuring languages (such as HTML and XML), meta-data languages (such as RDF), data transformation languages (such as XSL/T/FO), data presentation languages (such as CSS), server-side programming languages (such as PHP, JSP), and client-side programming languages (such as Javascript).

Building on these *Content Management Systems* (CMS for short) emerged as a technology providing a unified interface for large databases or data warehouses. However, they do not solve all the challenges in the area. Some of the persistent problems include distributed authoring, where number of users provide and modify the content, the difference between content and knowledge, where the system should not just store data. Some of these issues were partially resolved with the introduction of specialized CMS called *wikis*, that introduce a

[⋆] The paper is supported by the *HEKATE* Project funded from 2007–2009 resources for science as a research project.

highly distributed authoring model. This technology is being enhanced with the development of *semantic wikis* allowing for semantic annotation of the content. The next step towards true knowledge-based systems is provided by knowledge wikis, that introduce explicit knowledge representation.

In this paper a new approach to the knowledge wiki architecture is proposed. It is based on a simple yet flexible and powerful idea of incorporating Prolog language code into the content stored in the wiki system. In the paper the design of such a wiki, based on the DokuWiki is introduced and a prototype implementation is discussed.

The rest of the paper is organized as follows. In Sect. 2 a discussion of main features and limitations of CMS is contained. Next, in Sect. 3 the wiki systems are discussed. The main problem with classic CMS, including wikis, is the lack of support for true knowledge management, as pointed out in Sect. 4. Some Wiki extensions, such as semantic and knowledge wikis presented in Sect. 5, aim at overcoming it. A knowledge wiki can be built around the classic knowledge representation techniques used in the Prolog language (see Sect. 6). A Prolog based extension to a wiki system is proposed in Sect. 7, with a prototype implementation for the DokuWiki introduced in Sec. 8. Some examples for the prototype are shown in Sect. 9. The paper ends with directions for future work in Sect. 10.

2 Content Management Systems

In a broad sense a *Content Management System* (CMS) consists of: a RDBMS (Relational Data Base System) which stores the data, and a complex web-based application or interface providing the access to the data. The web interface is built around common web technologies. This software requires a proper and efficient run-time environment, including a web server. An important aspect of a CMS system is the possibility of user personalization, and interactive use.

Multiple CMS categories can be identified: web portals, groupware suites, forum sites, and e-learning toolkits. In each of these cases the content they manage differs. Several classes or groups can be identified. The main differentiating factor is the type of services they are oriented for. A special case of CMS is a *wiki*, described in the next section.

From a knowledge engineering and management perspective, content may be seen as a particular kind of knowledge. While all of the CMS systems provide some kind of content structuration, few of them focus on using proper knowledge representation methods. The gathered content is only human readable, and hardly subject to machine processing. This, in fact, exposes an important conceptual gap between content and knowledge management systems. Wheres the former simply store data, the latter should allow for a semantic-aware processing.

3 Wiki Systems

The wiki concept emerged in the 90's. The main idea was to create a simple and expressive tool for communication and knowledge sharing. A wiki system is mainly a collaboration tool. It allows multiple users to access, read, edit, upload and download documents. It has a regular client-server architecture. Documents are text-based, enriched with so-called wiki markup. It is a simplistic, tag-based, text only language which allows the user to annotate text with information regarding its structure and presentation. Such an enriched text is called *wikitext*. The tags allow users to make sections, subsections, tables, items and other typographic and structural operations. A wikitext remains human readable, tags are intuitive and easy to learn.

Each document is uniquely identified by a keyword, which makes the wiki concept similar to the encyclopedia concept. Furthermore a document, in addition to typographic tags mentioned earlier, can contain hyperlinks to other documents. A hyperlink is a document name enclosed by a link tag. The wiki allows users to upload images, as well as other files and link them together. One of the most important features of a wiki is an integrated version control. Each page modification is recorded. At any time a user can access any previous version of any page.

Wikis are mostly web-based. The web interface allows users to access, see and edit wiki pages. Depending on the particular solution, there might be access control and authorization mechanisms implemented. A wiki system is usually based on server side processing technologies providing the web application, and optionally a database back-end. As a front-end a web browser is used.

It is worth pointing out that wiki systems currently blend with regular Content Management Systems. Some CMS provide wiki functionality while some wikis evolve into CMS. Similarly wikis are more and more often merged with *e-Learning* systems to support collaborative knowledge gathering and sharing.

One of the most interesting wiki systems for developers is DokuWiki (`wiki.splitbrain.org/wiki:dokuwiki`). It is designed to be both easy to use and easy to set up. DokuWiki is based on PHP and does not require any database back-end. Pages are stored as versioned text files which enables easy backup-restore operations. It allows for image embedding, and file upload and download. Pages can be arranged into so-called namespaces which act as a tree-like hierarchy similar to directory structure. It also provides syntax highlighting for in-page embedded code for programming languages such as: C/C++, Java, Lisp, ADA, PHP, SQL and others, using GeSHi (`qbnz.com/highlighter`).

Furthermore, it supports extensive user authentication and authorization mechanisms including Access Control Lists (ACL). Its modularized architecture allows the user to extend DokuWiki with plugins which provide additional syntax and functionality. The most popular plugins provide: user and ACL management, blog, gallery of pictures, discussion board, calendar, LaTeX symbols rendering, and GraphViz visualization.

4 Content vs. Knowledge

A review of the selected CMS, including wikis reveals some common limitations; namely: technical limitations, content management and portability problems, and most importantly, oversimplified knowledge representation.

When it comes to the the actual management of the content the main problem is, that CMS systems provide limited content portability. As long as content is simply data, such as HTML, PDF documents, pictures and so-on, CMS acts as a repository. When it comes to sharing information about the content, meta-data, or meta-knowledge, CMS systems do not provide appropriate facilities. This is mainly related to the lack of some common knowledge representation standards, while focusing on low-level encoding, and visual presentation of content.

From a knowledge management point-of-view, the crucial problem with CMS systems is, that they escape the knowledge representation and management pitfalls by assuming some predefined CMS structure, e.g. a portal. In this sense a CMS is only a set of modifiable portal templates. It is difficult to choose knowledge representation for current CMS solutions, because the knowledge they store can be considered implicit, or even hidden.

This criticism implicitly assumes that what CMS were designed to manage, and make available, is somehow knowledge. That would put the process of deploying a CMS in the context of knowledge engineering. In this field concepts such as knowledge representation, management, or validation are used referring to common tasks. However, considering how most of CMS are built, they cannot be considered Knowledge Management Systems. Turning current CMS into knowledge-oriented CMS should involve an introduction of number of features.

There seems to be generic areas that should be supported by any "knowledge server"; these are: knowledge representation and organization support; the system should provide appropriate structures and languages, knowledge processing and inference; the system should be capable of multi-paradigm reasoning and automated operations on knowledge, inference and operation control, user support, and operational decision making, knowledge acquisition support and extraction of knowledge from different sources, contextual user interface - both for operational use and administration/design, truth-maintenance, verification and validation support, learning and optimization.

There are numerous efforts to meet these requirements. However, because of lack of consistent and uniform theoretical foundations, efforts oriented towards building a knowledge server replacing, subsuming and covering database services, web applications and decision processes, are far from being satisfactory. Recent developments in the area of intelligent web authoring and collaboration tools include the development of semantic wiki systems. They offer the functionality of semantic annotation of the content. The next stage is provided by knowledge wikis, which add explicit knowledge representation and processing capabilities with the introduction of decision rules and trees.

5 Semantic Knowledge Wikis

A first step in the direction of enriching standard wikis with the semantic information has been performed by the introduction of the so-called *semantic wikis*, such as the IkeWiki [1], or OntoWiki [2]. In such systems the standard wikitext is extended with the semantic annotation. Such annotations allow for building an ontology of the domain with which the content of the wiki is related. This extension introduces not just new content engineering possibilities, but also semantic search and analysis of the content. In order to provide annotations semantic wikis allow for RDF or OWL annotations.

However, from the knowledge engineering point of view expressing semantics is not enough. In fact a knowledge-based system should provide effective knowledge representation and processing methods. In order to extend semantic wikis to knowledge-based systems, the concept of *semantic knowledge wikis* has been introduced, see [3,4]. An example of such a system is the *KnowWE* semantic knowledge wiki [3,5]. In such a system the semantic knowledge is extended with the problem-solving domain-specific knowledge. The system allows for introducing knowledge expressed with decision rules and trees related to the domain ontology. So conceptually it is built on top of the semantic wikis.

The approach presented in this paper shares goals with the semantic knowledge wikis. However, it differs with respect to the methods used. In the paper a generic solution based on the use of Prolog as the language for expressing both the semantics, and the knowledge processing information is presented.

6 Prolog Knowledge Representation Model

The Prolog language [6,7] is a prime example of applied programming in logic, where knowledge representation and processing is critical. Knowledge is represented with use of facts and rules as Horn clauses in First-Order Predicate Calculus. This formal representation intuitively corresponds to statements in the natural language. The language offers powerful knowledge processing capabilities, including recursive processing, unification and resolution. The flexibility of this approach, and the extensibility of the language made it an AI tool of choice for many systems.

Prolog natively uses a backwards chaining reasoning strategy, that corresponds to logical abduction. However, thanks to its meta-programming facilities it is easy to build any Prolog-based interpreter for a custom reasoning, including forward chaining.

Knowledge is expressed as clauses being facts or rules, often referred to as the knowledge base. Clauses constitute predicates. The way how facts are stated is close to the natural language, with a verb and a set of nouns being predicate name and predicate arguments respectively. Let us assume that there are some facts regarding student project management: there is a student Frank, a teacher Bob, both of them work on project `knowledgeWiki`, and the project is already finished (`done`).

```
is_student(frank).
is_teacher(bob).
works_on_project(frank,knowledgeWiki).
works_on_project(bob,knowledgeWiki).
project_status(knowledgeWiki,done).
```

Rules are expressed in a similar way with use of Predicate Calculus which introduces a concept of variable (indicated as capitalized identifiers). Finding out which student needs to be graded can be formulated as:

```
need_grade(Who,Project):-
  is_student(Who), works_on_project(Who,Project),
  project_status(Project,done), \+ grade(Who,Project,_).
```

which can be read as: `Who` needs a grade from `Project` if `Who` is a student and he works on `Project`, and the project status is `done` and `Who` dos not have a grade assigned (`\+` means negation).

Combining facts and rules gives a very powerful and expressive technique for storing knowledge. It is far more capable than simple representations such as RDF, which defines facts only.

Furthermore, Prolog allows to specify a goal to the inference engine to actually query the knowledge base. To find out who works on the `knowledgeWiki` project a query could be issued:

```
?- works_on_project(Who,knowledgeWiki).
```

Prolog will try to unify a solution with `Who` variable giving the answer. Queries are just like complex clauses, they can be integrated with the knowledge base. The following displays on standard output who works on `knowledgeWiki`:

```
:- works_on_project(Who,knowledgeWiki), write(Who).
```

Prolog-based knowledge representation can be possibly far richer than the ones used in the current knowledge wikis implementations. In the next section a new proposal of a knowledge wiki is introduced. It allows for a direct integration of Prolog code with the wikitext.

7 Prolog Knowledge Wiki Proposal

A simple concept for a generic knowledge wiki is to integrate Prolog code into the wiki text. Such Prolog annotations or attachments would enhance the content with semantics, as well as provide an inference technique. This idea was first proposed in [8].

The basic functionality should at least include:

1. ability to include Prolog code into wikitext,
2. spawning the Prolog inference engine in order to interpret the code embedded in wiki pages,

3. possibility of analyzing wiki contents by the Prolog program.

This would ultimately mean that a bidirectional Wiki-to-Prolog interface has to be developed.

The first requirement seems to be easy to fulfill. Considering how wikitext is built, a special markup for the Prolog code can be provided. A function to interpret larger Prolog files included as files in the wiki can also be considered.

The second one needs some low-level runtime integration, where the web-server is able to spawn custom process. A Prolog interpreter would be run, with Prolog-related wiki contents provided.

It should be possible to access and analyze the contents of the wiki from within the Prolog code. A special library of predicates that access the wiki, using wiki-specific references should be provided.

Another issue are the actual use scenarios for the system. Several possibilities have to be taken under consideration:

- enhance wiki with some *meta-knowledge*,
- describe the *meaning* of wiki contents with Prolog,
- provide knowledge *processing* for the wiki.

Common vocabulary, predicate names, their arities, and meaning should be defined as well. It would give a common meaning to the knowledge gathered in the wiki forming an ontology. Applying OWL or RDFS at this point should be evaluated. The proposed enhancement addresses issues presented in Sect. 4.

8 System Implementation

A preliminary implementation of concepts described in the previous section has been carried out within two students projects in the Knowledge Engineering Methods class of 2008 at AGH UST and a master's thesis research, see [9]. The prototype is based on the DokuWiki system. The implementation uses a dedicated Prolog plugin for DokuWiki. The plugin meets requirements 1. and 2. (see Sect. 7), being able to interpret Prolog code embedded into wiki, as well as spawning the Prolog interpreter.

The Prolog code is included using a simple construct:

```
wikitext

<prolog>
color(red).
</prolog>

wikitext
```

This instructs the wiki, that the contents should be parsed by the Prolog plugin.

In order to run the code, thus spawn the Prolog interpreter, a Prolog goal should be explicitly stated with the plugin invocation, e.g.:

```
wikitext

<prolog goal="color(X),write(X),nl,fail.">
color(red).
color(green).
color(blue).
</prolog>

wikitext
```

Input output issues are resolved as follows. Knowledge gathered in the wiki is treated as input, there is no explicit user input of any kind (no forms, input fields etc.). This knowledge is subject to processing in the way specified above. Prolog standard output is rendered in the page in the place where the plugin invocation is present. Thus, the above example is seen as a wiki page as follows:

```
wikitext

red
green
blue

wikitext
```

since `write/1` generates variable value and `nl/0` a new line on standard output. In this very case `fail/0` forces backtracking, which makes `X` to be resolved with `red`, `green`, `blue` values in turn.

The interpreter can access selected areas of the wiki, using an explicit scope specification, such as:

```
<prolog goal="color(X),write(X),nl,fail."  scope="pl:miw">
<prolog goal="female(X),write(X),nl,fail." scope="pl:miw:proj.*08">
```

In the first case the contents of the whole wiki namespace[1] is parsed (all documents within `pl:miw` namespace). In the second case a number of pages matching the given regular expression are parsed (which are pages within `pl:miw` namespace which names start with `proj.` and end with `08`). The contents of all the pages are concatenated into a single Prolog file and then analyzed by the interpreter.

Another solution allows for using Prolog files stored in wiki, or outside it, accessible with an URL. This is especially useful in case of larger files e.g.:

```
<prolog goal="female(X),write(X),nl,fail" file="pl:miw:test.pl">
<prolog goal="female(X),write(X),nl,fail" url="http://sth.org/test.pl">
```

The current implementation is a proof-of-concept prototype. It does have some limitations, including:

[1] The concept of a namespace is typical to DokuWiki, however it is also present in other wiki systems, particular syntax indicating a namespace differs between systems.

- scope specification – it is not entirely compatible with the default DokuWiki namespace handling,
- full input/output from the Prolog program – currently it is limited to explicit output handling encoded in the `goal` specification for the plugin,
- flexible goal specification – in general one can use the `:-` expression anywhere in the Prolog file, to provide a goal, with the current version of the plugin this not always works as expected,
- caching – this poses some important problems related to both functionality as well as efficiency; in some cases it is desirable to override or disable the DokuWiki rendering cache to get results generated by the Prolog engine; on the other hand a separate caching mechanism for the Prolog code itself is provided, however, it does have certain limitations,
- unidirectional interface – currently it is only possible to invoke Prolog interpreter from the wiki; there is no way to actually query the wiki from Prolog,
- there is no syntax checking of Prolog code – it is possible to enter syntactically incorrect code without any notification about it,
- there is no editing support such as code completion.

See `https://ai.ia.agh.edu.pl/wiki/prolog:prologwiki` for project progress as well as testing area. There is also an ongoing master's thesis research carried out in this domain.

9 Use Example

The examples discussed here present some basic use for the plugin. They are also available online, see `https://ai.ia.agh.edu.pl/wiki/prolog:prologwiki`.

```
====== Smith Family ======
There is Kate and Liz obviously girls, and a couple of boys: Tom and Bob.
Kate is Bob's mom, while Tom is his dad.
Tom is also Liz's dad.
<prolog scope="family">
female(person(kate,smith)).
female(person(liz,smith)).
male(person(tom,smith)).
male(person(bob,smith)).
parent(person(kate,smith),person(bob,smith)).
parent(person(tom,smith),person(bob,smith)).
parent(person(tom,smith),person(liz,smith)).
</prolog>
```

Figure 1. `smith_family`: Smith family description

Expressing family relationships is a good example for using knowledge wiki. An example description of Smith Family as `smith_family` page, is given in Fig. 1.

It defines both a description of the family (clear wikitext) and corresponding, formal definition with use of `female/1`, `male/1` and `parent/2` predicates. Furthermore there is a scope defined, which interprets knowledge gathered in `family` page, see Fig. 2. These are additional, common rules allowing to infer family relationships such as maternity or paternity.

```
======  Family Definitions ======
===== Father and Mother =====
A parent and a female is somebody's mother.
A parent and a male is somebody's father.
<prolog>
mother(X,Y) :- parent(X,Y),female(X).
father(X,Y) :- parent(X,Y),male(X).
</prolog>
```

Figure 2. `family`: Common definitions regarding families

```
====== Families ======
Knowledge about all families is gathered in this namespace.
<prolog scope=".*_family">
</prolog>
```

Figure 3. `families`: Gathering knowledge about all families

There could be another page (`families`) which gathers knowledge about all families present in the wiki (within current namespace), see Fig. 3.

10 Perspectives for Future Work

This paper presents an original idea of implementing a semantic knowledge wiki using Prolog for knowledge representation and processing. In the paper a proof-of-concept prototype implementation for the DokuWiki system is described. Since the current prototype has number of limitations, there is an ongoing effort to extend this idea.

Embedding predicate logic in terms of Prolog code into a wiki system delivers a powerful tool for expressing and processing knowledge. However, there are some constraints needed to make knowledge exchangeable and reusable. A similar approach can be observed in the AceWiki [10] project (`http://attempto.ifi.uzh.ch/acewiki`). It provides a semantic wiki that uses a controlled natural language called ACE. Compared to most other semantic wikis it does not use RDF or OWL directly, the semantics is contained directly in the wiki text and not in form of annotations.

A vocabulary should be defined which establishes a relationship between predicates and their meaning. Knowing a list, probably structured, of predicates and their meanings, gives a basis for using, reusing and extending them. Similarly, there should be a clear mechanism for defining such relationships. Existing technologies such as ontologies should be considered, as well as these close to the Prolog language, such as self-documenting help mechanism. There are three directions visible now: an OWL or RDF based ontology, a wiki-based one, or a purely Prolog-based one.

Current version of the prototype does not introduce Prolog debugging features. It is not possible to check the syntax of the Prolog code, or assist the user in entering it. In the future both debugging and syntax highlighting features are planned. And extended interface that uses automatic hinting could also be provided.

There are obviously some performance issues regarding knowledge processing within a wiki system based on Prolog code interpretation. Extracting knowledge from many pages and processing it could be a time consuming operation. Some smart caching techniques should be used then. A research regarding this issue has already been started. It is based on the caching mechanism present in the DokuWiki system.

As it is described in Sect. 8 a goal for the inference engine could be specified at the `prolog` tag. This approach seems to be a little redundant since there is already a Prolog built in mechanism to state a goal, integrating it with a knowledge base, with use of `:-`, see Sect.6. Application of either one and their semantical differences will be researched.

Using scope attribute can lead to some ambiguity. If there are many `prolog` elements within a page with different scopes, a global scope of the page is a sum of all the scopes. Alternatively there could be no such a concept as the page global scope. In this case scopes will be applied on element basis. Alternatively a scope could be defined with use of a special predicate instead of an attribute.

The proposed Prolog-based knowledge wiki is not compatible with the semantic annotation mechanism found in semantic wikis. These include the use of technologies such as OWL or RDF. There should be some bridging interface allowing for such cooperation and knowledge exchange established.

One of the promising applications of the proposed technology is to provide a learning environment for students taking Artificial Intelligence courses. The knowledge wiki gives a chance to learn Prolog without using other technologies than a web browser. Furthermore, even complicated knowledge repositories with distributed facts and rules spanning over many wiki pages, with multiple users and cooperative user interaction, can be build this way. It suits very well demonstration purposes as gathered knowledge can be put to work immediately.

References

1. Schaffert, S.: Ikewiki: A semantic wiki for collaborative knowledge management. In: WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling

Technologies: Infrastructure for Collaborative Enterprises, Washington, DC, USA, IEEE Computer Society (2006) 388–396

2. Auer, S., Dietzold, S., Riechert, T.: Ontowiki - a tool for social, semantic collaboration. In Cruz, I.F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L., eds.: International Semantic Web Conference. Volume 4273 of Lecture Notes in Computer Science., Springer (2006) 736–749
3. Baumeister, J., Reutelshoefer, J., Puppe, F.: Knowwe: community-based knowledge capture with knowledge wikis. In: K-CAP '07: Proceedings of the 4th international conference on Knowledge capture, New York, NY, USA, ACM (2007) 189–190
4. Reutelshoefer, J., Baumeister, J., Puppe, F.: Ad-hoc knowledge engineering with semantic knowledge wikis. In: Proc. of SemWiki 2008 - The Wiki Way of Semantics, Workshop co-located with the 5th European Semantic Web Conference, Tenerife, Spain (2008)
5. Baumeister, J.; Puppe, F.: Web-based knowledge engineering using knowledge wikis. In: Proc. of the AAAI 2008 Spring Symposium on "Symbiotic Relationships between Semantic Web and Knowledge Engineering", Stanford University, USA (2008) 1–13
6. Bratko, I.: Prolog Programming for Artificial Intelligence. 3rd edn. Addison Wesley (2000)
7. Covington, M.A., Nute, D., Vellino, A.: Prolog programming in depth. Prentice-Hall (1996)
8. Nalepa, G.J., Wojnicki, I.: Concept of an interactive web portal for teaching prolog. In: FLAIRS2008. (2008) accepted.
9. Kotra, M.: Dokuwiki plugins for graphviz and prolog. Knowledge Engineering Project (MIW), G. J. Nalepa supervisor (2008)
10. Kuhn, T.: AceWiki: Collaborative Ontology Management in Controlled Natural Language. In: Proceedings of the 3rd Semantic Wiki Workshop, CEUR Workshop Proceedings (2008)