Chapter 1
Embedding the HeaRT Rule Engine Into a Semantic Wiki*

Grzegorz J. Nalepa, Szymon Bobek

Abstract Semantic wikis provide a flexible solution for collaborative knowledge engineering. The reasoning capabilities of most of the semantic wiki systems are limited to lightweight reasoning, e.g. with the use of RDF. Number of systems try to integrate strong reasoning with rules. Since most of the regular rule engines are hard to integrate, custom solutions are needed. In the paper a practical approach for embedding a flexible rule engine called HeaRT into a semantic wiki is given. HeaRT supports several inference modes, as well as rule base structuring, which makes it particularly suitable for a wiki.

1.1 Introduction

The Internet can be considered a biggest knowledge repository in the world. The amount of data available online is so huge, that searching it becomes a problem. The difficulties in finding relevant data are caused not by the lack of information, but by the great amount of unstructured data.

Due to this problems, systems that introduce structure and semantics to knowledge bases were developed. Semantic wikis [2] are one of the most popular Semantic Web systems that provide reasoning in such knowledge bases. Semantic wikis provide a simple formalism for semantically annotating its content, semantic search, and other manipulation of knowledge not available in regular wikis. Currently, many implementations of semantic wikis use mostly lightweight reasoning solutions based on RDF. Some more advanced systems include limited OWL support. However, there are limitations of these

AGH University of Science and Technology,
Al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: gjn@agh.edu.pl, szymon.bobek@agh.edu.pl

* The paper is supported by the BIMLOQ Project funded from 2010–2012 resources for science as a research project.

systems to perform rule-based reasoning required by more demanding knowledge engineering applications.

The original contribution of this paper is the proposal of combining a semantic wiki with a rule-based inference engine. The PlWiki [4] system is used, with the embedded HeaRT [8] rule engine. This is a flexible solution, that uses expressive rule language (XTT2 [9]). A rule-based recommender system case is considered.

The paper is organized as follows: In Sec. 1.2 important implementations of semantic wikis are described. The Sec. 1.3 contains problem definition and motivation for the solution presented in the following sections. In Sec. 1.4 the HeaRT inference engine is described. The Sec. 1.6 contains description of combining rule-based reasoning engine with a semantic wiki system. An use case example of this solution is shown in Sec. 1.7. Summary and future work are presented in Sec. 1.8.

1.2 State-of-the-Art in Semantic Wikis

In recent years there have been a number of system proposals that aim at introducing reasoning within knowledge bases. One of the most popular systems are semantic wikis that combine collaborative knowledge acquisition with semantic annotations. They allow for flexible knowledge processing and simple reasoning. In this section popular semantic wiki systems are described. The emphasis of the description is on the knowledge representation and reasoning.

IkeWiki[10] was one of the first semantic wiki allowing semantic annotation with RDF and OWL support and OWL-RDFS reasoning. Research on the IkeWiki has been continued by the Kiwi [11] project, where an effort was put mainly into improving collaborative knowledge management, with not so much emphasis on reasoning.

Semantic MediaWiki [2] is one of the most popular semantic wiki. It is built on the top of the MediaWiki adding possibilities to embed ontological annotation within the text of wiki articles, that are translated into OWL and can be queried by users with a wiki-like syntax.

One of the most interesting approaches to semantic wiki systems is represented by the AceWiki[3]. The system uses ACE (*Attempto Controlled English*) language to represent knowledge. The ACE language is processed by the AceWiki engine and can be mapped bidirectionally to OWL language. AceWiki also provides an AceRules module that allows for forward chaining inference in rule-based knowledge bases. Inference in such knowledge bases is based on first-order logic.

KnowWE[1] is a semantic wiki system that was build as an extension to JSP wiki system. It allows for heterogeneous semantic knowledge representation with use of decision trees, rules and set-covering models. Although the

rules representation is more powerful than in AceWiki, the inference strategies and capabilities to manage structured rule bases are still simple.

PlWiki [4] allows for adding ontological annotation within articles and provides reasoning capabilities using powerful Prolog resolution algorithm. Embedding Prolog code within wiki articles enables expressing knowledge in a rule-based way using Horn clauses. The Prolog source code can be added explicitly or by special *wiki-like* markups that are translated to Prolog terms. The PlWiki wiki system has been developed as a plugin to DokuWiki. In fact, PlWiki is a prototype of the general concept of Loki, a wiki that provides expressive knowledge representation with rules [6, 7].

1.3 Motivation

Rules are one of the most popular and powerful knowledge representations. However, practical reasoning with rules is not a trivial task. Thus, most of the semantic wikis do not support them for inference tasks. In fact, reasoning task in semantic wiki systems can be understood in two ways: 1) *lightweight reasoning* – including mostly classification problems, and 2) *strong reasoning* – a reasoning that involves rules and facts processing.

To improve the reasoning capabilities of semantic wikis, there is a need for providing strong reasoning. Existing classic rule engines (e.g. Drools, Jess, CLIPS) are in most cases too heavy and hard to integrate for this task. Therefore, semantic wikis providing inference engines use mostly lightweight reasoning. Recently some of the wiki systems have been integrating ontology reasoners. In fact, with the new OWL 2 RL Profile it is possible to use simple rule semantics with OWL. However, that current support for this feature in DL reasoners as well as semantic wikis using them, is very limited.

These limitations give motivation to propose a new system, combining a semantic wiki flexibility in collaborative knowledge engineering with the power of rule-based representation, allowing for advanced inference. Due to the fact that embedding a rule engine such as CLIPS, or Drools within a wiki system is difficult, a new solution should be provided. In the next section integration of the PlWiki system with the HeaRT rule engine is proposed.

1.4 HeaRT Inference Engine

HeKatE RunTime (HeaRT) [5] is a lightweight embeddable rule inference engine built as a part of the HeKatE project[2] [9]. The distinctive features of the HeaRT engine are the following:

[2] See `http://hekate.ia.agh.edu.pl`.

- support for an expressive rule language (XTT2) that has a complete formal definition in the ALSV(FD) logic [9],
- the use of modularized rule bases: rules working in the same context are clustered into decision units forming an inference network with advanced inference strategies including forward and backward chaining are provided.
- rule base verification mechanisms that allows for checking for logical completeness, and redundancy in the rule base, and
- lightweight and embeddable implementation using a fast Prolog compiler.

Knowledge in the XTT2 representation uses extended decision tables. The tables are connected between and create an inference network. An example of the decision tables network is presented in Fig. 1.1.

Visual representation of the rule base is human-readable, but difficult to parse and process. To address this issue the HMR language is used. It is a simple and readable text representation suitable for automated processing. The textual representation is automatically generated from the visual one, by a dedicated design tools [9]. HMR is the native rule language for the HeaRT rule engine.

The HMR language includes definitions of attributes and their types, rule and tables specification. An example excerpt of HMR and its interpretation is given below. This is a part of a basic movie recommendation system presented in Fig. 1.1. Using several attributes it tries to recommend a movie to a wiki user. The example includes two rules that depending on an age of the user and preferable movie genre, decides what types of movie are allowed for the person.

```
xattr [name: movie_types, type: genres, class: general].
xattr [name: age_filter,  type:age_selection, class:general].

xrule filter/1:
  [age lt 18, movie_types sim [horror, thriller]]  ==>
  [age_filter set []].
xrule filter/2:
  [age lt 18, movie_types sim [science-fiction]]   ==>
  [age_filter set union(age_filter,[young_sf])]:sf_rules.
```

The meaning of rules from given examples is as follows: If the age of the user is less than 18, and s/he selected that s/he would like to watch a horror, or thriller, then set the filter to empty set, and stop. If the age is less than 18 and s/he selected science-fiction movie, then set the filter to science-fiction movies for young and go to rules responsible for making a suggestion on the movie title.

Rules in the form presented above, are directly processed by HeaRT. The engine supports several inference modes. *DDI* (a data-driven forward inference) identifies start tables, and puts all tables that are linked to the initial ones in the table network into a FIFO queue. When there are no more tables to be added to the queue, the algorithm fires selected tables in the order

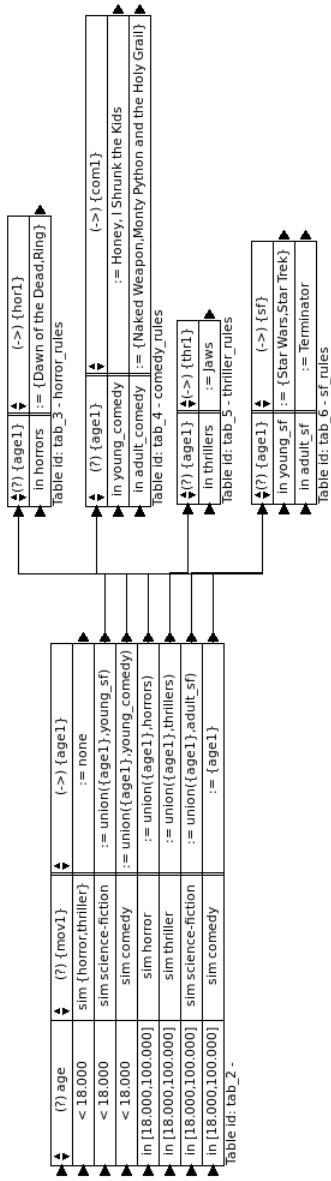

Fig. 1.1 XTT2 visual representation of simple movie recommendation system

they are popped from the queue. *GDI* (goal-driven backward chaining) works backwards with respect to selecting the tables necessary for a specific task, and then fires the tables forward to achieve the goal. One or more output

tables are identified as the ones that can generate the desired goal values and are put into a LIFO queue. As a consequence, only the tables that lead to the desired solution are fired, and no rules are fired without purpose. Moreover, a *TDI* (token-driven inference) suitable for complex inference networks is provided. This approach is based on monitoring the partial inference order defined by the design pattern structure with tokens assigned to tables. A table can be fired only when there is a token at each input. Intuitively, a token at the input is a flag, signalling that the necessary data generated by the preceding table is ready for use.

The engine is a stand alone application that can be easily integrated with a design environment, or a runtime framework. In fact, it can also be easily embedded into another application. The current implantation uses a fast and portable Prolog interpreter and compiler (SWI-Prolog[3]) which makes it easy to deploy.

1.5 Introduction to PlWiki

The main objectives of PlWiki are to enhance both representation and inference features, allow for a complete rule framework in the wiki. A decision has been made to build the new wiki with use of the Prolog language. This allows to provide rich knowledge representation, including rules, as well as allow for an efficient and flexible reasoning in the wiki, and expressive power equivalent to Horn clauses. Thus it is possible the represent the domain specific knowledge and reasoning procedures with the same generic representation. The system provides a semantic layer, as well as the Semantic Media Wiki compatibility features.

PlWiki was built on the top of the DokuWiki. It is a simple, popular and fast wiki engine mainly for creating documentation, and storing information on-line. DokuWiki pages are stored on the server as text files and later parsed by the wiki engine which renders XHTML. DokuWiki architecture is extensible with extensions called plugins.

The current version of PlWiki implements the Syntax and Renderer plugin functionality. Text-based wikipages are fed to a lexical analyzer (Lexer) which identifies the special wiki markup. In PlWiki the standard DokuWiki markup is extended by a special `<pl>...</pl>` markup that contains Prolog clauses. The stream of tokens is then passed to the Helper plugin that transforms it to special renderer instructions that are parsed by the Parser. The final stage is the Renderer, responsible for creating a client-visible output (e.g. XHTML). In this stage the second part of the PlWiki plugin is used for running the Prolog interpreter and invoke HeaRT.

Below basic use examples of the generic Prolog representation are given:

[3] See `http://swi-prolog.org`.

```
<pl>movie_title('Terminator','StarWars').
    movie_type(science-fiction).</pl>
```

This simple statement adds two facts to the knowledge base. The plugin invocation is performed using the predefined syntax. To actually specify the goal (query) for the interpreter the following syntax is used:
`<pl goal="movie_title(X),write(X),nl,fail"></pl>`
It is possible to specify a given scope of the query (with wiki namespaces):

```
<pl goal="movie_title(X),movie_type(X),fail"
    scope="prolog:examples"></pl>
```

A bidirectional interface, allowing to query the wiki contents from the Prolog code is also available, e.g.:
`<pl goal="consult('lib/plugins/prolog/plwiki.pl'),`
`wikiconsult('plwiki/pluginapi'),list."></pl>`.

There are several options how to analyze the wiki knowledge base (that is Prolog files built and extracted from wiki pages). A basic approach is to combine all clauses. More advanced uses allow to select pages (e.g. given namespace) that are to be analyzed. On top of the basic Prolog syntax, semantic enhancements are possible. These are mapped to Prolog clauses for processing.

1.6 Embedding the HeaRT Engine in the Wiki

In this Section integration of powerful rule-based inference engine (HeaRT) with wiki engine (PlWiki) is proposed. HeaRT inference engine is written in Prolog, so it can be run using PlWiki. HMR language that is used to represent rule-based knowledge in HeaRT is also interpreted directly by Prolog and can be embedded on wiki pages as well. HMR language supports knowledge modularization which is inevitable in wiki systems, where information is spread over many pages or namespaces. HeaRT inference engine takes advantages of this modularization providing advanced inference strategies (see Sec. 1.4).

In Fig. 1.2 the architecture of PlWiki with HeaRT is presented. It is divided into two modules: the first module is responsible for rendering wiki pages, and extracting the HMR code, the second module is embedded within PlWiki engine and it is responsible for performing inference based on the HMR model passed to it by the PlWiki engine.

The process of rendering a wiki page in PlWiki with HeaRT looks as follows:

1. Wiki engine parses the wiki page and extracts HMR code and reasoning queries (goals) for HeaRT.
2. Depending on a scope defined in the goal, PlWiki merges the HMR code from wiki pages in a given scope and passes it to HeaRT.

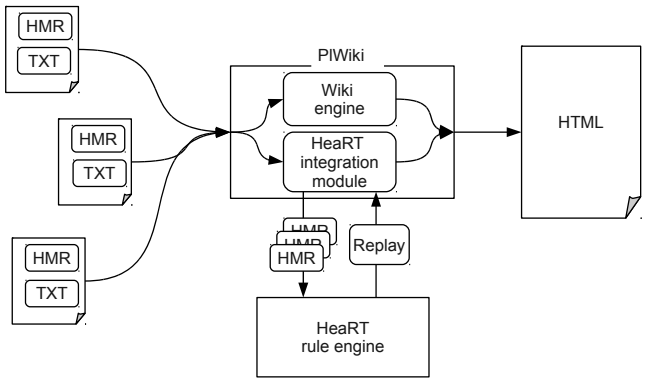

Fig. 1.2 Architecture of PlWiki and HeaRT system

3. HeaRT performs the reasoning process and returns results to PlWiki engine.
4. PlWiki renders complete wiki page with previously parsed regular text and an answer to a given query (goal) produced by HeaRT.

HeaRT inference engine was added to the PlWiki as a part of a plugin responsible for parsing Prolog. HMR language is embedded on wiki pages with the `<pl></pl>` tag. To run reasoning a `<pl scope="" goal="">` tag is used. If the goal is valid HeaRT command for running inference process, then the reasoning is performed by the engine, result calculated and rendered on a wiki page.

To run inference in HeaRT, the *gox* command is used that takes three parameters: values of input attributes, rules to be process, and reasoning mode. Values of input attributes are passed as a name of the state element of HMR language. The state element stores values of attributes values. Rules that have the same attributes in conditional part and the same attributes in decision part are grouped in one table. Therefore, to pass to HeaRT rules that has to be processed, in fact names of the tables that contain them should be passed. An example of running invoking an inference in PlWiki is:

```
<pl scope="*" goal="gox(init,[result_table],gdi">
```

The meaning of the example is: run the Goal-Driven inference treating *result_table* as a goal table and taking values of input attributes from the state called *init*. The *scope* parameter in *<pl>* tag is optional and it specifies a namespace from which types, attributes, tables and rules should be taken as an input for reasoning process. If not specified, the entire knowledge in wiki is processed. The *goal* parameter is mandatory, and it has to be a valid HeaRT command.

1.7 Use Example

The simple movie recommendation system presented in this section tries to recommend a movie set for a user of a given age and some film genre preferences. It uses separate namespaces for each user of the system and additional namespace for movie list. The XTT2 diagram containing rules and reasoning flow in the system is presented on Fig. 1.1.

Fig. 1.3 Reasoning results on user page

In first step the system decides for which subsets of movies the user is allowed based on his age. For instance user that age is below 18, is not allow to watch horrors and thrillers, nor other movies that age limit is higher than 18. Secondly, based on this age filter, the system search for movies that best fit user preferences specified in his profile. At the end, the system response containing the list of recommended movies is printed on the PlWiki page. To initialize attribute values representing user age, and preferences, the *xstat* element from HMR language is used (See Fig. 1.4). In Fig. 1.3 a sample output from the system is presented.

Rules that correspond to the age filter are located in *movies* namespace. Rules responsible for matching movies to user preferences and the age filter are located on separate pages, finally the user profile is located in the user personal namespace. An example of rules, written in HMR language, that matches movies to user prefferences are shown below:

```
xrule filter/1:
  [age lt 18, movie_types sim [horror, thriller]]
  ==> [age_filter set [none]].
xrule filter/2:
```

```
    [age lt 18, movie_types sim [science-fiction]]
    ==> [age_filter set union(age_filter,[young_sf])]
    :sf_rules.
  xrule filter/3:
    [age lt 18, movie_types sim [comedy]]
    ==> [age_filter set union(age_filter,[young_comedy])]
    :comedy_rules.
  xrule filter/4:
    [age in [18 to 100], movie_types sim [comedy]]
    ==> [age_filter set union(age_filter,[adoult_comedy])]
    :comedy_rules.
  xrule filter/5:
    [age in [18 to 100], movie_types sim [horror]]
    ==> [age_filter set union(age_filter,[horrors])]
    :horror_rules.
  xrule filter/6:
    [age in [18 to 100], movie_types sim [thriller]]
    ==> [age_filter set union(age_filter,[thrillers])]
    :thriller_rules.
  xrule filter/7:
    [age in [18 to 100], movie_types sim [science-fiction]]
    ==> [age_filter set union(age_filter,[adoult_sf])]
    :sf_rules.
```

To merge all this information, a scope has to be given when goal for the inference is specified. Scope accepts POSIX Regular Expressions, so to collect the HMR model located in several namespaces, the construction presented in Fig. 1.4 is used.

It is also possible to browse through the database of movies, and get information whether selected movie is recommended for the user or not. What is important, this functionality can be achived without modyfying a rule base. Running different inference mode (in this case Goal-Driven Inference, which is an implementation of backward chaining in HeaRT) will test if selected movie is one would be recommended for the user. Only one line of code is reuired to enable this functionality.

An example below shows how to make a rule engine answer question whether there are comedies movied recommendation tor the user:

```
<pl scope="[user|movies]"
    goal="gox(user,[comedy_rules],gdi), print_results."></pl>
```

It is worth noting that this modularization of HMR model gives an opportunity to extend the system easily for other product recommendation (e.g. books). It would require from the user (or system developer) to change the scope parameter in the goal tag from *movies* to *books* and the system would use user profile data to make a suggestion on which books best match the user preferences.

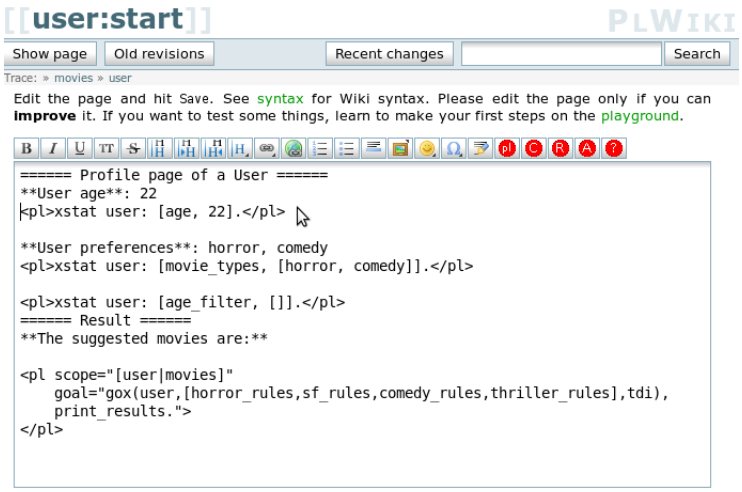

Fig. 1.4 Goal query on user profile page

1.8 Summary and Future Work

In the paper a system that combines a semantic wiki with the power of rule-based inference engines is presented. It allows for complex rule representation, knowledge base structuring, and advanced inference strategies. Presented solution is a prototype, implementing the following functionality: creating wiki articles with embedded Prolog code, and rule-based knowledge, organizing rules into modules and design inference flow within the knowledge using HMR language, querying system that performs reasoning on the knowledge and produces answers that are later rendered on the article page, and performing partial logical verification of the knowledge.

The PlWiki with HeaRT prototype is oriented towards practical applications in community sites, where individual users work together in a collaborative manner. Rule-based decision modules in a wiki can also be useful in e-commerce applications.

The functionality does not include any user interface that helps in creating knowledge, and querying the system. The complete knowledge of HMR language syntax is required. It is inconvenient for the random user even though the language is relatively human readable and intuitive. What is more current solution requires from user to manually organize rules into contexts (tables). In future it should be done automatically, so that the user would have to concentrate only on writing a rules.

References

1. 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, pp. 189–190. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1298406.1298448
2. Krötzsch, M., Vrandecic, D., Völkel, M., Haller, H., Studer, R.: Semantic wikipedia. Web Semantics **5**, 251–261 (2007)
3. Kuhn, T.: AceWiki: A Natural and Expressive Semantic Wiki. In: Proceedings of Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges. CEUR Workshop Proceedings (2008)
4. Nalepa, G.J.: PlWiki – a generic semantic wiki architecture. In: N.T. Nguyen, R. Kowalczyk, S.M. Chen (eds.) Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, First International Conference, ICCCI 2009, Wroclaw, Poland, October 5-7, 2009. Proceedings, *Lecture Notes in Computer Science*, vol. 5796, pp. 345–356. Springer (2009)
5. Nalepa, G.J.: Architecture of the HeaRT hybrid rule engine. In: L. Rutkowski, [et al.] (eds.) Artificial Intelligence and Soft Computing: 10th International Conference, ICAISC 2010: Zakopane, Poland, June 13–17, 2010, Pt. II, *Lecture Notes in Artificial Intelligence*, vol. 6114, pp. 598–605. Springer (2010)
6. Nalepa, G.J.: Collective knowledge engineering with semantic wikis. Journal of Universal Computer Science **16**(7), 1006–1023 (2010)
7. Nalepa, G.J.: Loki – semantic wiki with logical knowledge representation. In: N.T. Nguyen (ed.) Transactions on Computational Collective Intelligence III, *Lecture Notes in Computer Science*, vol. 6560, pp. 96–114. Springer (2011)
8. Nalepa, G.J., Bobek, S., Gawędzki, M., Ligęza, A.: HeaRT Hybrid XTT2 rule engine design and implementation. Tech. Rep. CSLTR 4/2009, AGH University of Science and Technology (2009)
9. Nalepa, G.J., Ligęza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science **20**(1), 35–53 (2010)
10. 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, pp. 388–396. IEEE Computer Society, Washington, DC, USA (2006). DOI http://dx.doi.org/10.1109/WETICE.2006.46
11. Schaffert, S., Eder, J., Grünwald, S., Kurz, T., Radulescu, M.: Kiwi – a platform for semantic social software (demonstration). In: ESWC, pp. 888–892 (2009)