KnowWE – A Wiki for Knowledge Base Development

Joachim Baumeister[1] **, Jochen Reutelshoefer**[1]**, Volker Belli**[1]**,**
Albrecht Striffler[1]**, Reinhard Hatko**[2] and **Markus Friedrich**[1]

Abstract. The development of knowledge systems has been driven by changing approaches, starting with special purpose languages in the 1960s that evolved later to dedicated editors and environments. Nowadays, tools for the collaborative creation and maintenance of knowledge became attractive. Such tools allow for the work on knowledge even for distributed panels of experts and for knowledge at different formalization levels. The paper (tool presentation) introduces the semantic wiki KnowWE, a collaborative platform for the acquisition and use of different types of knowledge, ranging from semantically annotated text to strong problem-solving knowledge. We also report on some current use cases of KnowWE.

1 Introduction

The utility of decision-support systems proved in numerous examples over the past years. The actual progression of knowledge-based systems goes back to the early years of expert systems. Starting with dedicated AI languages, such as LISP [22] and Prolog [15], task-driven tools have been developed to construct intelligent systems more efficiently, e.g., see [6, 7]. Recently, a number of development tools promoted the creation of knowledge on different formalization levels. That way, explicit process knowledge (e.g., rules, decision trees, fault models) can be linked with ontological relations or even text and multimedia content. Semantic wikis [21] are a prominent example for supporting such a *knowledge formalization continuum* [3], e.g., see the systems Semantic MediaWiki[14], PlWiki [16], and MoKi [8].

In this paper, we introduce the semantic wiki KnowWE that emphasizes the development of strong problem-solving knowledge within the knowledge formalization continuum. The system is the latest successor of a 30-years list of ancestors of diagnostic expert shell kits. Starting with the system MED1 [19] and MED2 [17] (initially implemented in INTERLISP, then ported to FRANZLISP) the knowledge engineers needed to use an internal knowledge representation syntax to built the knowledge bases. The successor D3 [18]—an implementation in Allegro Common Lisp—offered a graphical user interface based on forms, tables, and trees to simplify the knowledge acquisition and to enable domain specialists to define the knowledge by themselves. The full reimplementation d3web (started in 2000 and implemented in Java) brought multi-user and multi-session capabilities to the reasoning engines and also offered a web-based user interface for developed knowledge bases for the first time. As well, the knowledge modeling environment KnowME (Knowledge Modeling Environment) was implemented in Java and copied the graphical editors of the shell-kit D3, but also added sophisticated tools for testing and refactoring the developed knowledge bases [5].

However, all aforementioned systems only support the work of one knowledge engineer at the same time, thus hindering a collaborative and distributed development process with many participants. Furthermore, the graphical editors restricted the structuring possibilities of the knowledge bases by the system-defined structure and expressiveness. In consequence, the engineers often needed to fit their knowledge structure into the possibilities of the tool. More importantly, the mix of different formalization levels was not possible, e.g., by relating ontological knowledge with solutions of a decision tree.

As the successor of KnowME the system KnowWE (Knowledge Wiki Environment) [4] offered a web-based wiki front-end for the knowledge acquisition and supported the collaborative engineering of knowledge at different formalization levels. Strong problem-solving knowledge is mixed with corresponding text and multimedia in a natural manner. The knowledge base can be flexibly structured by distributing the particular knowledge modules over a collection of linked wiki articles, each covering a particular aspect of the domain.

In the following sections, we describe notable features and developments of the system KnowWE and we briefly discuss some current applications.

2 Applications and Usage of KnowWE

In this section, we first sketch the typical application domains of KnowWE and then we describe typical practices for knowledge development with the system.

2.1 Application Domains

Historically, the typical use of the system was the development of diagnostic knowledge bases, since this problem category was the core domain of d3web and its predecessors. Nowadays, KnowWE is still used to develop decision-support systems for diagnosis, classification, or recommendation tasks. As KnowWE can be also used for ontology engineering and clinical guideline engineering, however, the application areas are broadened today. For example, we see applications for the definition of clinical guidelines [9], the configuration of HCI devices [13], and the ontological formalization of ancient history [20].

In summary, almost all applications combine formal knowledge with informal content of the wiki, thus improving the development and the use of the knowledge system. In the following section we describe basic practices for developing knowledge bases with KnowWE.

[1] denkbares GmbH, Friedrich-Bergius-Ring 15, D-97076 Würzburg, Germany, email: name.surname@denkbares.com

[2] Department of Intelligent Systems, University of Würzburg, Germany, email: name.surname@uni-wuerzburg.de

2.2 Practices for Knowledge Development

Distribution of Knowledge In form-based tools the knowledge is typically entered in predefined editor fields. That way, the knowledge engineer is bound to the given organization strategy of the particular tool. In a semantic wiki the engineer is free to partition and distribute the knowledge across the wiki articles. Thus, specific articles can be created to define the particular aspects of the knowledge base. In many cases, this freedom is a significant advantage when compared to form-based tools, since the distribution strategy can be adapted to the current project requirements and the characteristics of the knowledge. However, in any way the knowledge engineer has the burden to formulate a distribution strategy for the knowledge in the wiki before starting with the knowledge engineering task.

In the past, a number of useful distribution patterns have been identified. It is important to notice that the patterns can/should be modified according to the project requirements, and that they can be combined with other patterns.

- **Solution-oriented distribution**: For each possible system output (or coherent group of outputs), an article is created in the wiki. The article contains the definitions of the output and formal knowledge to derive this particular output. For larger systems, sub-articles can be defined that are linked from the main article.
- **Problem area-oriented distribution**: For each problem area (coherent and named groups of inputs to the system), an article is created in the wiki. Each article contains the definitions of the problem area (e.g., symptoms concerning the problem area) and links to articles, where derivation knowledge is defined relevant to the particular problem areas.
- **Concept-oriented distribution**: For each concept of the application domain an article is created. Attributes and relations of this concept are also defined on this article. Also links to related concepts are included.

Namespaces and Compilation of Knowledge In the past, tools only allowed the creation of one knowledge base at the same time. Current environments enable the development of a collection of knowledge bases within one workspace. Here, coherent parts of knowledge need to be clustered and labeled by namespaces. For smaller knowledge bases, namespaces are often used to tag the knowledge relevant for this knowledge base.

A dedicated article is used as a sink for the definition of a knowledge base, i.e., to collect the knowledge packages for the specified namespaces. That way, a wiki can be used to create different variants of a knowledge base, i.e., by having an article compiling all knowledge labeled with namespaces *n1*, and by having another article compiling all knowledge labeled with namespaces *n1* and *n2*. The namespaces and corresponding compilation of knowledge is depicted in Figure 1.

As a historical remark, the current mechanism of namespaces and their compilation is different from the original ideas of KnowWE described in [4]: Back then, every article was compiled into a single knowledge base and therefore had to include all relevant concepts and derivations. In a distributed problem-solving process the different wiki articles and knowledge bases, respectively, communicated with each other exchanging input and output concepts. The outputs of the problem-solving process were displayed to the user in an aggregated view. The concept of distributed problem-solving uncovered two critical issues in real-world knowledge base development: First, the reasoning process was not intuitive for domain specialists who were usually not familiar with distributed reasoning algorithms. To help the users, very sophisticated explanations for derived solutions needed to be presented in order to allow for effective debugging when problems appeared. Second, the wiki often was used only as the *development environment* of the knowledge base. The target platform of the knowledge system typically differed from the wiki system, so the knowledge base needed to be joined and exported from the wiki into a *single knowledge base* to be applicable for the later use. In consequence, the exported knowledge base needed further quality management, since the reasoning results of the distributed reasoning may differed from the reasoning results of the monolithic knowledge base. Therefore, the test and development of a monolithic knowledge base (the setting of the target platform) within the wiki appeared to be more efficient for developers.

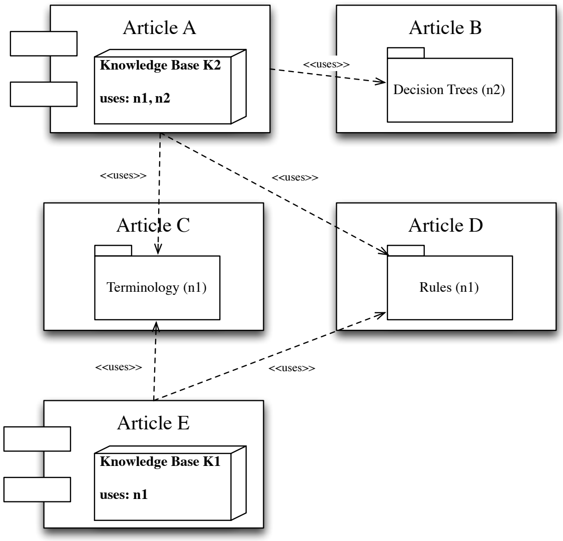


Figure 1. Distribution and namespaces of the knowledge across a set of wiki articles. Knowledge bases are flexibly compiled be defining a number of namespaces.

Endpoints for Testing the Knowledge During knowledge base development it is important to have powerful interfaces to test the current state of the knowledge base. In KnowWE, we offer a dialog interface for testing strong problem-solving knowledge, i.e., by presenting a form to enter values for input concepts. Derived solutions are presented in a configurable output panel. The test dialog and output panel can be placed in an arbitrary wiki article in order to give the user the required flexibility to test the knowledge base where it is currently developed.

For ontology engineering we offer a markup to formulate SPARQL [24] queries for RDF ontologies [23]. For OWL ontologies we are able to formulate specific class expression queries in Manchester OWL syntax [12].

Simple Support for Authoring Administration Within a collaborative development process not all involved engineers are working on the knowledge base at the same time. Moreover, the engineers are often not located at the same place. Therefore, the tool needs to offer support for administrative authoring tasks. Typical examples are as follows:

- Label unfinished areas of the knowledge base, i.e., todo tasks.

- Mark identified issues in knowledge definitions, i.e., problems.
- Specify urgent tasks for the development phase.

For all these tasks a specific user or group of users needs to be attachable in order to personalize them.

KnowWE offers a simple todo markup, that can be used to label content or formal knowledge in the wiki article with the action requests as described above. Furthermore, a tagging plugin allows for the annotation of entire pages. Tag clouds with instant access to tagged pages can be inserted into the wiki; most often at the bottom of the left navigation panel.

Use of Standard Wiki Features KnowWE benefits from a set of useful features, that usually comes with a standard wiki distribution:

- A user and group management allows for the fine-grained definition of user rights (view and edit) for single articles.
- All wiki articles are under version control. That way, older versions of an article (and its contained knowledge definitions) can be compared with the current version of the article. When necessary an older version of an article can be restored.
- A *recent changes* view displays a list of recently modified articles and knowledge definitions. With this feature, it is easy to keep track of the current development process.

3 Notable Features

KnowWE is a development environment that supports the knowledge engineer on all aspects of the development process, such as authoring assistance, error handling, refactoring, manual testing, and quality management. In this section we present a selection of the most relevant features of KnowWE.

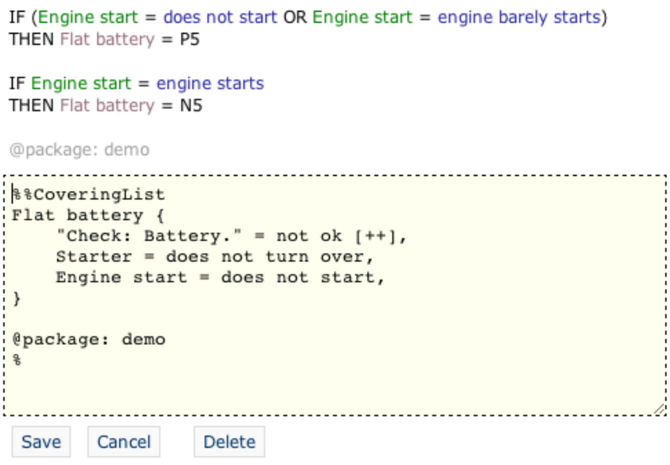
3.1 Knowledge Acquisition

In KnowWE, knowledge is formalized by using (knowledge) markup languages. A markup language is a formal syntax provided with an internal mapping to the target knowledge representation which is performed instantly after page save by a compilation script. The markup languages can be used at any place in the wiki articles to create elements of the knowledge base allowing for interweaving formal and informal knowledge. Figure 2 shows an article taken from an exemplary car fault diagnosis wiki describing the concept *Clogged air filter*. The article contains informal content such as plain text and images (e.g., in the top half of the article) as well as formalized knowledge (rules at the bottom part of the article). KnowWE provides markup languages for creating knowledge bases in the d3web[3] format and for creating ontologies in OWL. For the d3web reasoner, markups for decision trees, set-covering models, decision tables, and rules are provided as introduced in [2]. Additionally, executable flowcharts can be designed in the DiaFlux language by using a graphical editor available the wiki [10]. For the development of ontologies KnowWE provides markups based on well-known languages such as the Manchester Syntax for OWL [12] and the Turtle Syntax for RDF [4].

3.2 Authoring Support

In addition to the basic wiki editing interface, KnowWE provides different kinds of editing support. The system provides *instant edit* functionality that allows to edit a section, i.e. a coherent part of an article, within the view of the wiki page as shown in Figure 3.

Typically, the editing of tables is difficult when using the standard text markup for tables. Therefore, KnowWE provides instant editing capabilities for tables in a WYSIWYG style allowing each cell to be edited by one click as shown in Figure 5. The table content is stored within the wiki page source in standard wiki markup.



```
IF (Engine start = does not start OR Engine start = engine barely starts)
THEN Flat battery = P5

IF Engine start = engine starts
THEN Flat battery = N5

@package: demo
```

```
%%CoveringList
Flat battery {
    "Check: Battery." = not ok [++],
    Starter = does not turn over,
    Engine start = does not start,
}

@package: demo
%
```

Figure 3. Authoring parts of an article using the instant edit feature.

Additionally, a code completion mechanism supports the user to create markup sections in the text editing panel.

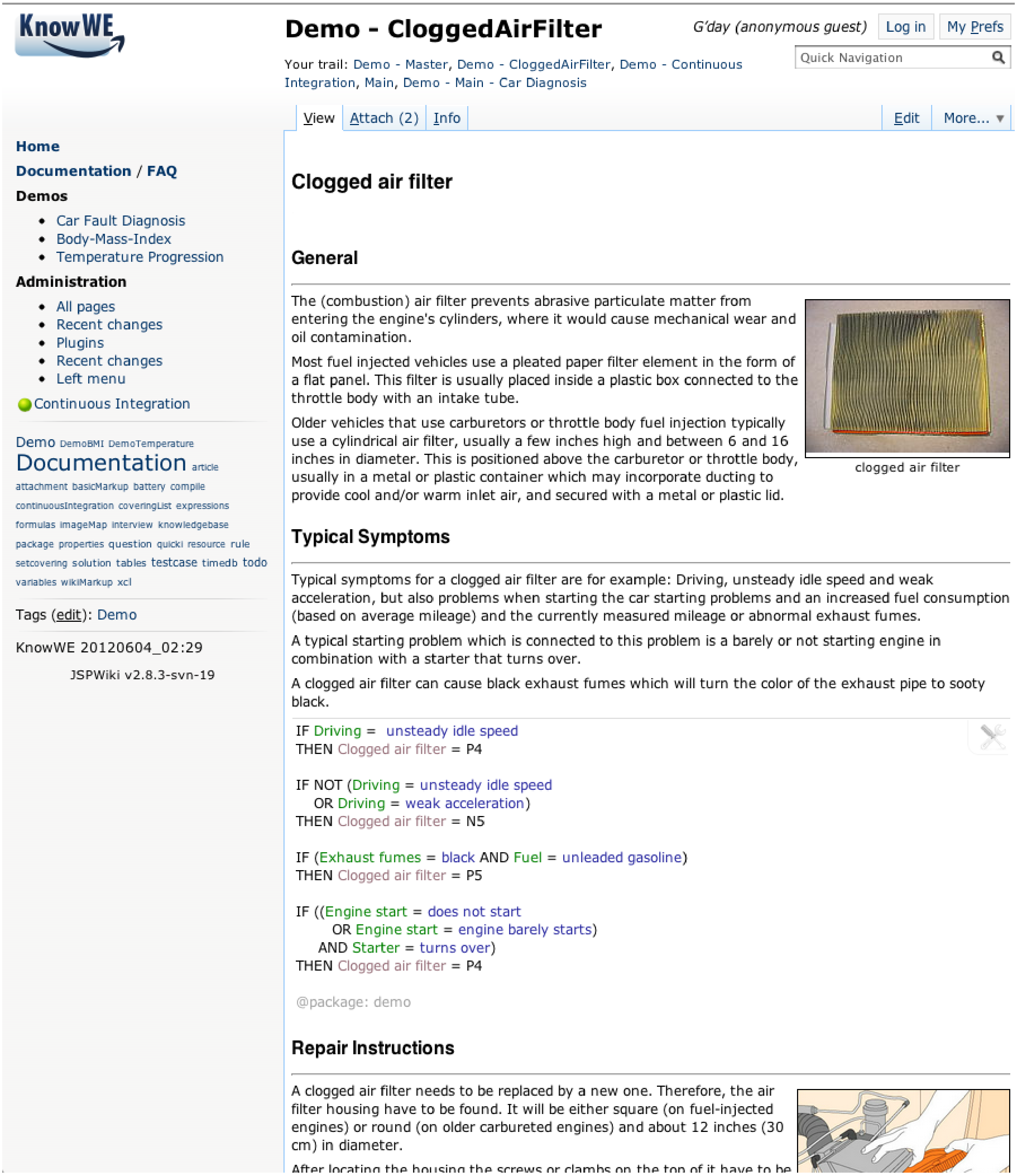
Often, it becomes necessary to obtain an overview of the occurrences and uses of a particular domain concept. Figure 4 shows an overview page for the concept *Leaking air intake system*, that is dynamically generated when requested by clicking on the concept name in the wiki. Besides the pure information about the concept, also small refactoring capabilities are available: At the top, a renaming tool is presented that allows the wiki-wide renaming of the concept, thus ensuring a working and consistent knowledge base. In the bottom part of the info page, the user can see an overview of the wiki articles, where the concept is used (links yield to the particular occurrences in the wiki).

3.3 Testing

As a modern knowledge engineering environment, KnowWE supports an agile knowledge engineering approach. Here, knowledge bases are developed in an evolutionary manner, always maintaining an executable and correct version at a certain level of competency. In this context, (automated) testing is very important to ensure successful evolutionary development cycles. Test cases are either developed manually by defining expected solutions for a given set of inputs or are imported from external testing suites. We adopted the continuous integration practice known from software engineering into the knowledge engineering tool KnowWE. A continuous integration dashboard in the wiki is used to define a collection of quality tests (for validation and verification). As a special knowledge markup, the dashboard can be configured easily to support tailored quality management for the respective project. Registered automated tests are performed on the current version of the wiki knowledge base and

[3] http://d3web.sourceforge.net

[4] http://www.w3.org/TeamSubmission/turtle


KnowWE
Demo - CloggedAirFilter
G'day (anonymous guest)
Log in
My Prefs
Quick Navigation
Your trail: Demo - Master, Demo - CloggedAirFilter, Demo - Continuous Integration, Main, Demo - Main - Car Diagnosis
View
Attach (2)
Info
Edit
More...
Home
Documentation / FAQ
Demos
Car Fault Diagnosis
Body-Mass-Index
Temperature Progression
Administration
All pages
Recent changes
Plugins
Recent changes
Left menu
Continuous Integration
Demo DemoBMI DemoTemperature Documentation article attachment basicMarkup battery compile continuousIntegration coveringList expressions formulas imageMap interview knowledgebase package properties question quickI resource rule setcovering solution tables testcase timedb todo variables wikiMarkup xcl
Tags (edit): Demo
KnowWE 20120604_02:29
JSPWiki v2.8.3-svn-19
clogged air filter

Clogged air filter

General

The (combustion) air filter prevents abrasive particulate matter from entering the engine's cylinders, where it would cause mechanical wear and oil contamination.

Most fuel injected vehicles use a pleated paper filter element in the form of a flat panel. This filter is usually placed inside a plastic box connected to the throttle body with an intake tube.

Older vehicles that use carburetors or throttle body fuel injection typically use a cylindrical air filter, usually a few inches high and between 6 and 16 inches in diameter. This is positioned above the carburetor or throttle body, usually in a metal or plastic container which may incorporate ducting to provide cool and/or warm inlet air, and secured with a metal or plastic lid.

Typical Symptoms

Typical symptoms for a clogged air filter are for example: Driving, unsteady idle speed and weak acceleration, but also problems when starting the car starting problems and an increased fuel consumption (based on average mileage) and the currently measured mileage or abnormal exhaust fumes.

A typical starting problem which is connected to this problem is a barely or not starting engine in combination with a starter that turns over.

A clogged air filter can cause black exhaust fumes which will turn the color of the exhaust pipe to sooty black.

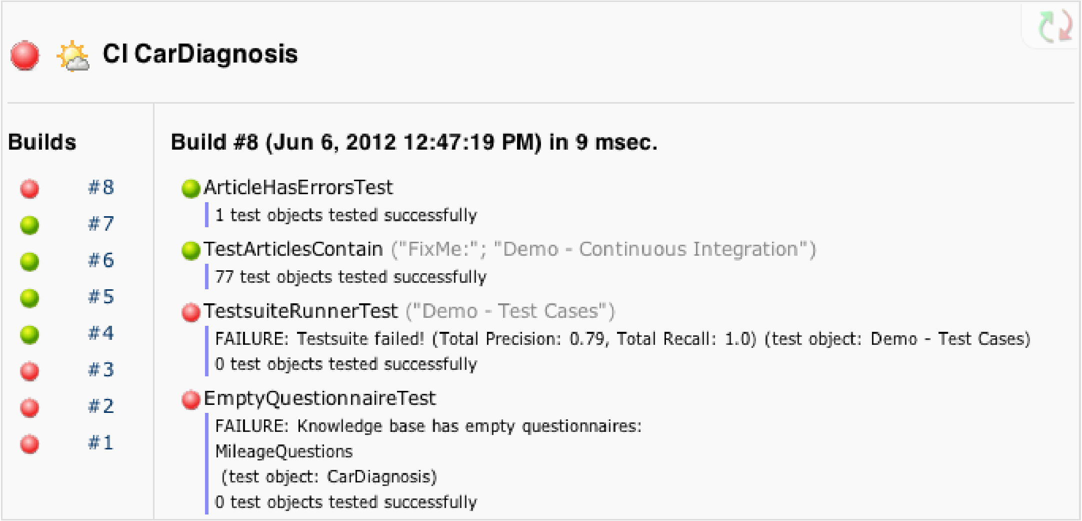
```
IF Driving =  unsteady idle speed
THEN Clogged air filter = P4

IF NOT (Driving = unsteady idle speed
    OR Driving = weak acceleration)
THEN Clogged air filter = N5

IF (Exhaust fumes = black AND Fuel = unleaded gasoline)
THEN Clogged air filter = P5

IF ((Engine start = does not start
        OR Engine start = engine barely starts)
     AND Starter = turns over)
THEN Clogged air filter = P4

@package: demo
```

Repair Instructions

A clogged air filter needs to be replaced by a new one. Therefore, the air filter housing have to be found. It will be either square (on fuel-injected engines) or round (on older carbureted engines) and about 12 inches (30 cm) in diameter.

After locating the housing the screws or clambs on the top of it have to be

Figure 2. A wiki page from a car-fault diagnosis knowledge base in KnowWE.



● CI CarDiagnosis

Builds

●	#8
●	#7
●	#6
●	#5
●	#4
●	#3
●	#2
●	#1

Build #8 (Jun 6, 2012 12:47:19 PM) in 9 msec.

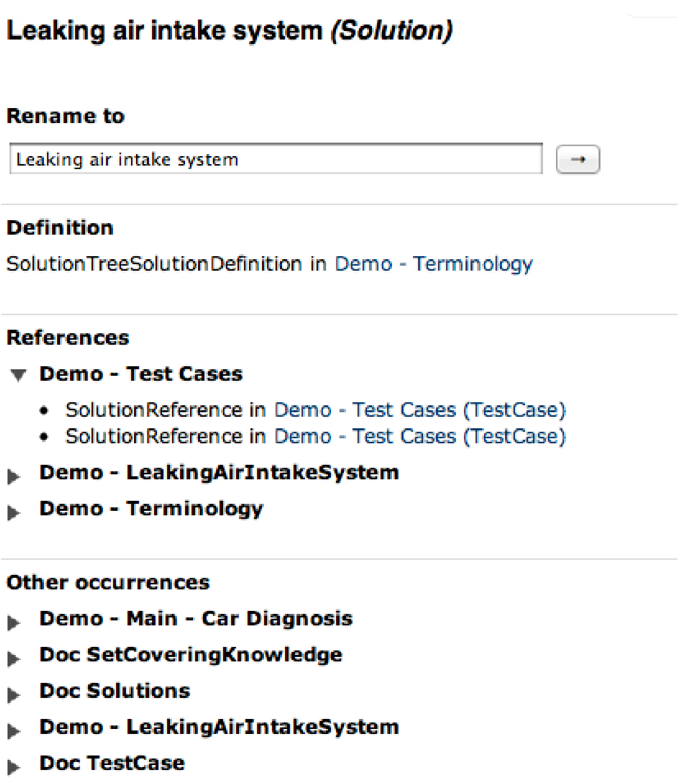
● ArticleHasErrorsTest
1 test objects tested successfully

● TestArticlesContain ("FixMe:"; "Demo - Continuous Integration")
77 test objects tested successfully

● TestsuiteRunnerTest ("Demo - Test Cases")
FAILURE: Testsuite failed! (Total Precision: 0.79, Total Recall: 1.0) (test object: Demo - Test Cases)
0 test objects tested successfully

● EmptyQuestionnaireTest
FAILURE: Knowledge base has empty questionnaires:
MileageQuestions
(test object: CarDiagnosis)
0 test objects tested successfully

Figure 6. The continuous integration dashboard of KnowWE showing messages of the current test runs and the history of the previous development stages.



Leaking air intake system *(Solution)*

Rename to

Leaking air intake system

Definition

SolutionTreeSolutionDefinition in Demo - Terminology

References

▼ Demo - Test Cases
- SolutionReference in Demo - Test Cases (TestCase)
- SolutionReference in Demo - Test Cases (TestCase)

▶ Demo - LeakingAirIntakeSystem

▶ Demo - Terminology

Other occurrences

▶ Demo - Main - Car Diagnosis

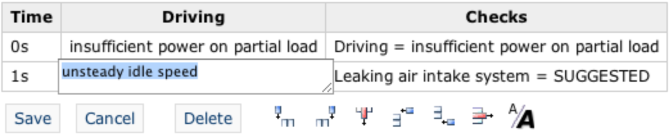
▶ Doc SetCoveringKnowledge

▶ Doc Solutions

▶ Demo - LeakingAirIntakeSystem

▶ Doc TestCase

Figure 4. The generated object-info page for every concept allows for the renaming of the concept and it shows the use of the concept across the wiki articles.



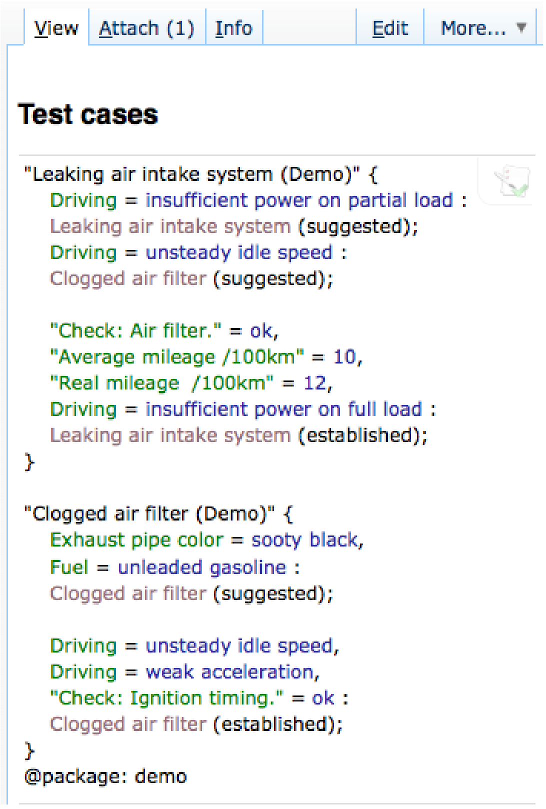
Time	Driving	Checks
0s	insufficient power on partial load	Driving = insufficient power on partial load
1s	unsteady idle speed	Leaking air intake system = SUGGESTED

Figure 5. Inline editing of tables by the WYSIWYG interface of the wiki.

give verbose feedback to the knowledge engineers by status messages on the dashboard as shown in Figure 6.

At any time, the dashboard displays the current state of the wiki knowledge base with respect to quality at one glance. Also the history of builds is listed on the left panel of the dashboard. Older builds can be inspected by clicking on the build number, for instance, because the developer wants to check the reason for the build problem. For the selected build the applied tests are shown in the center of the dashboard. In case of errors, the tests give detailed reports on the error s as well as links are provided for further investigation and debugging of the issue. In Figure 6, the top two tests have been passed successfully, while the lower two tests have failed showing more details explaining the actual problem. The tests can be activated by three trigger-modes *onChange*, *onSchedule*, and *onDemand*. In the mode onChange, the tests are executed after each modification of a wiki article which changed the knowledge base. This mode provides the most immediate feedback possible. However, for very time consuming tests this mode can yield inconvenient delays. The mode onSchedule executes the tests on a regular basis according to a specified schedule, for instance at night. This mode is preferable also for tests with considerable high execution time. Further, in the mode onDemand all responsibility for test execution is left to the user, since the user has to explicitly start a continuous integration run. The user has to decide, when the execution is reasonable, which often is an option for tests with high runtime (considering sufficiently experienced users). It is important to note, that the user can define different dashboards, for instance, one for quick tests running onChange and another one for executing larger/time-consuming tests onSchedule.

Additionally to the dashboard, located on a specific wiki page, KnowWE provides a *CI-Daemon* (daemon for continuous integration) which can be connected to a dashboard. The CI-Daemon is always visible in the KnowWE user interface basically only showing a colored bubble (green, red, or grey) representing the current state of the connected dashboard. In Figure 2 the CI-Daemon is visible as a green bubble on the left of the page below the navigation menu. In this way, the users are always aware of the current quality state not requiring to frequently visit the dashboard article. A very important category of tests for knowledge bases are the competency tests which can be implemented by (sequential) test cases [1]. Figure 7 shows a markup for the definition of sequential test cases in KnowWE. During execution, the test case is performed line-by-line. Equal signs express assignments of input data, added to the current testing session. Expressions containing brackets are expected derivations. The test fails, if the expected derivations do not match the actual ones. That way, input-output behavior of a knowledge base can be covered by automated competency tests which can be attached to a continuous integration dashboard easily.



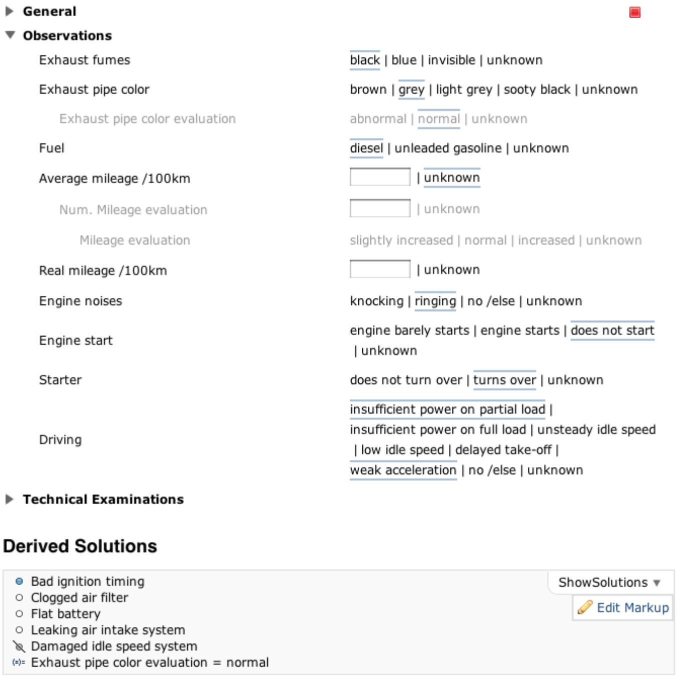
Test cases

```
"Leaking air intake system (Demo)" {
    Driving = insufficient power on partial load :
    Leaking air intake system (suggested);
    Driving = unsteady idle speed :
    Clogged air filter (suggested);

    "Check: Air filter." = ok,
    "Average mileage /100km" = 10,
    "Real mileage  /100km" = 12,
    Driving = insufficient power on full load :
    Leaking air intake system (established);
}

"Clogged air filter (Demo)" {
    Exhaust pipe color = sooty black,
    Fuel = unleaded gasoline :
    Clogged air filter (suggested);

    Driving = unsteady idle speed,
    Driving = weak acceleration,
    "Check: Ignition timing." = ok :
    Clogged air filter (established);
}
@package: demo
```

Figure 7. Markup for the definition of sequential test cases.

3.4 Knowledge Use

For instant manual testing of the created knowledge base KnowWE provides an embedded interview component which can be embedded into any wiki article. Figure 8 shows the interview interface which is dynamically generated from the connected knowledge base. It allows the user to answer the input questions and instantly gives feedback of the derived solution concepts. In the shown example, the combination of inputs derived the established solution concept *Bad ignition timing*. The solutions *Clogged air filter*, *Flat battery*, and *Leaking air intake system* are also suggested as potential solutions while *Damaged idle speed system* is marked as an excluded solution.



▶ General

▼ Observations

Exhaust fumes
[✓ black] | blue | invisible | unknown

Exhaust pipe color
brown | [✓ grey] | light grey | sooty black | unknown

Exhaust pipe color evaluation
abnormal | [✓ normal] | unknown

Fuel
[✓ diesel] | unleaded gasoline | unknown

Average mileage /100km
□ | [✓ unknown]

Num. Mileage evaluation
□ | unknown

Mileage evaluation
slightly increased | normal | increased | unknown

Real mileage /100km
□ | unknown

Engine noises
knocking | [✓ ringing] | no /else | unknown

Engine start
engine barely starts | engine starts | [✓ does not start] | unknown

Starter
does not turn over | [✓ turns over] | unknown

Driving
[✓ insufficient power on partial load] | insufficient power on full load | unsteady idle speed | low idle speed | delayed take-off | [✓ weak acceleration] | no /else | unknown

▶ Technical Examinations

Derived Solutions

● Bad ignition timing
○ Clogged air filter
○ Flat battery
○ Leaking air intake system
Damaged idle speed system
Exhaust pipe color evaluation = normal

Figure 8. The interview component for manual knowledge base testing.

For developed ontologies KnowWE provides an inline-query mechanism to summarize the knowledge of the ontology as a dynamic content element. Using a markup based on the SPARQL language, queries can be defined within the wiki pages. They are evaluated on page load on the current version of the developed ontology. The result of the query is displayed in the view of the wiki article.

4 Known Uses of KnowWE

KnowWE is currently used in several knowledge engineering projects of different subject domains, both in academic and industrial contexts. In this section, we report on a selection of these projects and we give a brief overview of the use of the system KnowWE.

4.1 Managing Chemical Safety with KnowSEC

KnowSEC (Managing Knowledge of Substances of Ecological Concern) is a group-wide wiki to manage substance-related work(flows) within a group of the German Federal Environmental Agency (Umweltbundesamt). Here, every substance is represented by a distinct wiki article storing important information such as chemical endpoints, relevant literature, or comments of group members. The information is entered in (user-friendly) editors in the wiki and translated into special markups in the background; thus, the information is also stored in an RDF ontology. That way, the information currently available in the wiki but also the latest knowledge changes can be aggregated and visualized by integrated SPARQL queries.

Besides the storage of weakly formalized knowledge, KnowSEC also offers knowledge-based modules that support the classification of substances for a number of critical chemical characteristics. At the moment, modules are available for supporting the assessment of the relevance, the persistence, the bioaccumulation, and the toxicity of a given substance. These aspects (e.g., relevance, persistence, etc.) are developed in the wiki using different namespaces, so they can be maintained and tested independently from the other aspects. For the users of KnowSEC, a joint knowledge base with all aspects is virtually defined including all above namespaces.

Currently, the knowledge base is still under development. The joint version of the knowledge base consists of 214 questions (user inputs to characterize the investigated substance) grouped by 46 questionnaires, 146 solutions (assessments of the investigated substance), and more than 1.000 rules to derive the assessments. The rules are automatically generated from entered decision tables that allow for an intuitive and maintainable knowledge development process.

Two knowledge engineers are supporting a team of domain specialists, that partly define the knowledge base themselves, partly giving domain knowledge to the knowledge engineers.

4.2 Modeling Clinical Guidelines in KnowWE

Within the project CliWE[5] (Clinical Wiki Environments), KnowWE is extended by plugins to allow for the collaborative development of Computer-Interpretable Guidelines (CIGs). Clinical guidelines are based on evidence-based medicine and improve patient outcome by providing standardized treatments. Their computerization allows for decision-support systems at the point of care, or even the automated application by closed-loop systems in the setting of Intensive Care Units. The goal of CliWE is to create a platform that supports the engineering of CIGs by spatially distributed domain specialists. Therefore, the graphical CIG language DiaFlux was created. Its focus lies on the direct applicability and understandability by domain specialists [9]. By offering only a small set of intuitive language elements, the guidelines can in the best case be built and maintained by the domain specialists themselves. Currently, the extensions developed within CliWE are used in the project WiM-Vent[6]. Its goal is to integrate medical expertise concerning mechanical ventilation and physiological models into an automated mechanical ventilator [11]. In the course of this project, one knowledge engineer guides and supports one domain specialists (backed up by a committee of further experts) during the knowledge engineering process. The latest version of the guideline contains 17 DiaFlux modules, that in total contain 295 nodes and 345 edges. During its development, the testing capabilities of KnowWE are extensively used. So far, about 1.100 continuous integration builds were automatically executed. Especially the empirical testing feature is applied to define and process local test cases, as well as ones that are created using external tools, e.g., a Human Patient Simulator. Those simulated patient sessions can then be replayed in KnowWE for introspecting and debugging the guideline execution. A high-lighting of the taken paths within the DiaFlux models serves as an accessible means of explanation for the domain specialists.

4.3 ESAT: Selecting Assisting Technologies for Handicaped People

ESAT (Expertensystem für Assistierende Technologien [german]) is an expert system designed to determine an appropriate set of human-computer interaction devices for handicapped people. In the application scenario a detailed profile of the physical capabilities (e.g., visual or motorical abilities) for a person is entered into the system. The knowledge base derives a set of input and output devices, that together provide optimal computer interaction for that specific person. In advance, the underlying domain knowledge has been elaborated by a comprehensive study in 2008. The actual implementation of a corresponding executable knowledge base using KnowWE has started in spring 2011. Currently, the ESAT knowledge base has been completed and the system will be launched for a testing phase at the project's initiator (FAB[7]). The knowledge base has been implemented by mainly one knowledge engineer using KnowWE. For knowledge representation production rules are used. In total the ESAT knowledge base currently contains 654 rules distributed on 74 wiki articles. Also in this single-user context the possibility of free structuring allows for reasonable and clear distribution of the knowledge. The terminology is defined on different wiki articles dealing with vision, hearing, motoric and haptic abilities and general skills (e.g., braille) respectively. The about 50 different types of input and output devices (e.g., various kinds of keyboards, sensors, displays) are each described in distinct wiki articles also containing the rules relevant for the derivation of the particular device. Five heuristics have been established within a theoretical study, describing solutions for major categories of handicaps. These are implemented on distinct wiki articles forming the core of the derivation knowledge. The testing framework for continuous integration discussed in Section 3.3 is extensively used to guarantee the save development process by uncovering undesired side-effects of modifications including at least one sequential test case for each device and heuristic. More details about the project are given by Kreutzer [13].

4.4 Continuous Medical Cataract Knowledge with WISSKONT

The WISSKONT project considers the creation of an intelligent information system in the medical domain of cataract surgery. The system is currently under development and it will support the ophtalmologist during the treatment process before, in-between, and after the cataract surgery. That way, the system needs to present relevant knowledge of the domain, which is integrated at varying degrees of formality. For instance, textbook content with images describe particular aspects of a treatment process, whereas temporal relations of the treatment phases are represented by ontological annotations. Here, informal content is correlated by ontological relations. In consequence, a semantic search mechanism provides the presentation of the relevant information at any stage of the treatment process. Additionally, for a number of decision tasks occurring during the treatment, distinct decision-support modules are created, e.g., the selection of an appropriate lens for the surgery based on the patient's parameters. The integration of formalized and informal knowledge allows the ophtalmologist to verify the recommendations of the knowledge base by analyzing the comprehensive support information provided with the recommendation.

The WISSKONT project is part of the WISSASS project, a cooperation of the Karlsruhe Institute of Technology, Germany (KIT) and

[5] funded by Draegerwerk AG & Co. KGaA, Lübeck, Germny, 2009-2012

[6] ”WiM-Vent” - Knowledge- and model-based Ventilation, funded by BMBF (Federal ministry of education and research)

[7] http://www.vo-fab.at/

the denkbares GmbH. It is funded as a ZIM-KOOP[8] project by the German Federal Ministry of Economics and Technology (BMWI).

5 Conclusion

In this paper, we presented the current state-of-the-art of the semantic wiki KnowWE. The tool is used in knowledge engineering projects that have a distributed and collaborative nature. Also, KnowWE is capable to jointly represent and use knowledge at different levels of formalization and therefore allows for the flexible organization and elicitation of knowledge. We showed notable features of the tool, such as dedicated markups and editors for knowledge acquisition and use, but also features for (continuously) testing the developed knowledge base. Publicly known projects and applications were reported, that use KnowWE as their primary knowledge engineering environment.

REFERENCES

[1] Joachim Baumeister, 'Advanced empirical testing', *Knowledge-Based Systems*, **24**(1), 83–94, (2011).

[2] Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe, 'Markups for knowledge wikis', in *SAAKM'07: Proceedings of the Semantic Authoring, Annotation and Knowledge Markup Workshop*, pp. 7–14, Whistler, Canada, (2007).

[3] Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe, 'Engineering intelligent systems on the knowledge formalization continuum', *International Journal of Applied Mathematics and Computer Science (AMCS)*, **21**(1), (2011).

[4] Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe, 'KnowWE: A semantic wiki for knowledge engineering', *Applied Intelligence*, **35**(3), 323–344, (2011).

[5] Joachim Baumeister, Dietmar Seipel, and Frank Puppe, 'Agile development of rule systems', in *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, eds., Giurca, Gasevic, and Taveter, IGI Publishing, (2009).

[6] B.G. Buchanan and E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, 1984.

[7] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu, 'The evolution of protégé: An environment for knowledge-based systems development', *Int. J. Hum.-Comput. Stud.*, **58**(1), 89–123, (January 2003).

[8] Chiara Ghidini, Barbara Kump, Stefanie N. Lindstaedt, Nahid Mahbub, Viktoria Pammer, Marco Rospocher, and Luciano Serafini, 'MoKi: The enterprise modelling wiki', in *ESWC'09: The Semantic Web: Research and Applications*, volume 5554 of *LNCS*, pp. 831–835. Springer, (2009).

[9] Reinhard Hatko, Joachim Baumeister, Volker Belli, and Frank Puppe, 'DiaFlux: A graphical language for computer-interpretable guidelines', in *KR4HC'11: Proceedings of the 3th International Workshop on Knowledge Representation for Health Care*, (2011).

[10] Reinhard Hatko, Jochen Reutelshoefer, Joachim Baumeister, and Frank Puppe, 'Modelling of diagnostic guideline knowledge in semantic wikis', in *Proceedings of the Workshop on Open Knowledge Models (OKM-2010) at the 17th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, (2010).

[11] Reinhard Hatko, Dirk Schädler, Stefan Mersmann, Joachim Baumeister, Norbert Weiler, and Frank Puppe, 'Implementing an automated ventilation guideline using the semantic wiki knowwe', in *EKAW 2012: 18th International Conference on Knowledge Engineering and Knowledge Management*, eds., Heiner Stuckenschmidt, Annette ten Teije, and Johanna Voelker, (2012).

[12] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H Wang, 'The manchester owl syntax', in *Proceedings of OWL: Experiences and Directions (OWLED'06)*, eds., Bernardo Cuenca Grau, Pascal Hitzler, Connor Shankey, and Evan Wallace, Athens, Georgia, USA,, (2006).

[13] S. Kreutzer, *Ein Expertensystem zur Unterstützung körperbehinderter Menschen*, Diplomica, 2012.

[14] Markus Krötzsch, Denny Vrandecić, and Max Völkel, 'Semantic MediaWiki', in *ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273*, pp. 935–942, Berlin, (2006). Springer.

[15] Dennis Merritt, *Building Expert Systems in Prolog*, Springer, Berlin, 1989.

[16] Grzegorz J. Nalepa, 'PlWiki - a generic semantic wiki architecture', in *ICCCI'09: Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, number 5796 in LNCS, pp. 345–356. Springer, (2009).

[17] Frank Puppe, 'Requirements for a Classification Expert System Shell and their Realization in MED2', *Applied Artificial Intelligence*, **1**, 163–171, (1987).

[18] Frank Puppe, 'Knowledge Reuse among Diagnostic Problem-Solving Methods in the Shell-Kit D3', *International Journal of Human-Computer Studies*, **49**, 627–649, (1998).

[19] Frank Puppe and Bernhard Puppe, 'Overview on MED1: An heuristic diagnostics system with an efficient control structure', in *Proceedings of the German Workshop on Artificial Intelligence (GWAI-83), Informatik-Fachberichte 76*, pp. 11–20. Springer, (1983).

[20] Jochen Reutelshoefer, Florian Lemmerich, Joachim Baumeister, Jorit Wintjes, and Lorenz Haas, 'Taking OWL to Athens – Semantic Web technology takes ancient greek history to students', in *ESWC'10: Proceedings of the 7th Extended Semantic Web Conference*, pp. 333–347. Springer, (2010).

[21] Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel, 'Semantic wikis', *IEEE Software*, **25**(4), 8–11, (2008).

[22] Guy L. Steele and Richard P. Gabriel, 'The evolution of lisp', in *The second ACM SIGPLAN conference on History of programming languages*, pp. 231–270, (1993).

[23] W3C. RDF - resource description framework recommendation: http://www.w3.org/rdf/, February 2004.

[24] W3C. SPARQL recommendation: http://www.w3.org/tr/rdf-sparql-query, January 2008.

[8] http://www.zim-bmwi.de/