Applied Intelligence, Springer, http://dx.doi.org/10.1007/s10489-010-0224-5
Preprint: The final publication is available at springerlink.com

KnowWE: A Semantic Wiki for Knowledge Engineering

Joachim Baumeister · Jochen Reutelshoefer · Frank Puppe

Received: 2009-11-24 / Accepted: 2010-02-25

Abstract Recently, Semantic Wikis showed reasonable success as collaboration platforms in the context of social semantic applications. In this paper, we present a novel approach, that interprets the concept of Semantic Wikis as a knowledge engineering environment, that effectively help to build decision-support systems. We introduce the Semantic Wiki KnowWE, that provides the possibility to define and maintain ontologies together with strong problem-solving knowledge. Thus, the wiki can be used to collaboratively build decision-support systems. These enhancements require extensions of the standard Semantic Wiki architecture by a task ontology for problem-solving and an adapted reasoning process. We discuss these extensions in detail, and we describe a case study in the field of medical emergency systems.

Keywords knowledge acquisition · knowledge engineering tools · decision-support systems

1 Introduction

In the last decades, the application of intelligent decision-support systems showed their advantages in many domains—examples of successful uses are described in the literature [13, 26, 28, 33, 35]. When building such systems, the most critical challenge is the development and maintenance of the knowledge bases. In the past, this challenge has been primarily tackled by the introduction of comprehensive methodologies describing the structured construction and application of the knowledge; examples are CommonKADS [50], the On-To-Knowledge Methodology [54], DILIGENT [55], and the Agile Methodology [12]. Corresponding tools are often tailored to the specific methodologies, and they usually limit the developer to a specific knowledge representation to be applied when building the system, for example Protégé [39, 22], OntoEdit [53], and KnowME [2, 4].

Today's knowledge engineering projects, however, often face the challenge that knowledge is present at different levels of formalization. Knowledge appears in different representations ranging from technical documents, construction plans, sheets, and experiences of human experts, but also in the explicit form of rules and models. Moreover, we see that

University of Würzburg, Institute of Computer Science, 97074 Würzburg, Germany
/joba,reutelshoefer,puppe/@informatik.uni-wuerzburg.de

the problem of knowledge formalization to one specific representation has not been solved sufficiently, i.e., the *knowledge acquisition bottleneck* still exists today.

1.1 Dilemmas of Knowledge Engineering

In our opinion and experience over the last years many projects failed because of conflicting options, that we call *knowledge engineering dilemmas*:

1. *The Single/Multiple Experts Dilemma.* The motivation and sophistication of *single domain specialists* is often the driving force of successful knowledge acquisition and evolution. Although high-quality experts can guarantee the construction of a high-quality knowledge base, these persons are often short in time and endurance. The distribution of the workload over a number of specialists would decrease this problem, but would also increase the risk of reducing the overall quality of the formalized knowledge.
   Furthermore, the collaboration among a group of specialists is not supported sufficiently in many (industrial) systems concerning the distributed development of a knowledge base. Here, the dilemma exists of favoring a distributed over a monolithic development process—involving multiple experts instead of a single expert.
2. *The Flexibility/Productivity Dilemma.* Current state–of–the–art tools are often tailored to a specific knowledge representation and acquisition interface for developing the knowledge base. In consequence, these tools are not sufficiently flexible to map the mental model of the domain specialists, that are responsible for formalizing the knowledge in the project. Additionally, *knowledge* appears in diverse representations, such as textual and tabular data, but also, for example, as explicit rules.
   On the one hand, the mapping of the particular mental model of the specialists to the provided knowledge representation and interfaces, respectively, often turned out to be difficult and time-consuming. On the other hand, a tool having the maximal flexibility, regarding the user interfaces and provided knowledge representations, typically would increase the complexity of its use and therefore decreases the productivity of the developers; this principle was also described in general as the *Flexibility-Usability Tradeoff* [32, p. 86]. In consequence, we face the dilemma of demanding a tool with maximal flexibility vs. a tool with maximal productivity.

Certainly, these dilemmas cannot be easily solved, but lightened by the introduction of agile and extensible tools, that adapt to the present situation. We motivate, that an extensible Semantic Wiki is an appropriate basis for building a new generation of knowledge engineering environments. It allows for the integration of knowledge at different levels of formality, and therefore tries to weaken the *flexibility/productivity dilemma* described above. The use of a Semantic Wiki additionally helps to target the first dilemma — the *single/multiple experts dilemma*. Due to its open and web-based implementation, a Semantic Wiki naturally allows for the distribution of the development process over a group of domain specialists. Collaboration is supported by many standard features of wikis, for instance distributed editing, versioning, rights management, and discussion pages. However, the dilemmas can only be lightened by providing a technical platform for a collaborative engineering process. Thus, a Semantic Wiki can be easily used within one of the methodologies mentioned above by serving as the primary development tool. For instance, in CommonKADS the documentation of the collected models can be naturally integrated into the system. In collaborative methodologies, such as DILIGENT, the Semantic Wiki can be used to support the agreements and discussions about the development process.

In this paper, we propose the Semantic Wiki KnowWE as a knowledge engineering environment for decision-support systems. The wiki is extended by the possibility to capture and share strong problem-solving methods for the classification task. Thus, it not only provides interfaces for the engineering of ontologies, but also interfaces for more expressive knowledge such as rules and fault models.

The rest of the paper is organized as follows: In Section 2, we introduce the wiki KnowWE in more detail: We show its functional organization, and we motivate how strong problem-solving knowledge is integrated into the ontological layer of a Semantic Wiki. We also briefly describe the reasoning architecture of KnowWE. One distinguishing component of KnowWE (in comparison to other Semantic Wikis) is that it provides textual markups to describe strong problem-solving knowledge for the classification task. Section 3 shows useful markups of KnowWE for the definition of rules, decision trees, and set-covering knowledge in more detail. The system is already used in industrial and scientific projects. We demonstrate the possible use of KnowWE in Section 5 by showing the development of a commercial medical decision-support system. The work is concluded with a discussion in Section 6.

2 Wikis for Knowledge Engineering

In the last years, wiki systems have shown their benefits as simple and versatile web-based content–management systems; users can add and modify tacit knowledge in form of text and multimedia in a flexible manner. As the most prominent example, Wikipedia attracts a large number of users, that are willing to create and maintain informal "world knowledge" through the wiki system. While wikis demonstrated their benefits for creating and sharing knowledge in open web environments, they are also successfully used in companies and universities as general knowledge management tools. The *key feature* of a wiki is its ability to change and refine content in a fairly simple way: Every wiki article is presented in a web browser in the corresponding *view mode*. The user can easily modify/extend the content of the article by changing into the *edit mode* of the article, which is usually possible due to a mandatory *Edit* button placed on the page. After saving the modifications, the changes are directly updated in the *view mode*.

Due to their simplicity, standard wiki systems show limitations when actually using the included knowledge. For knowledge retrieval, only a simple full-text search is available, and knowledge connected across different articles cannot be aggregated in a unified manner. This motivated the development of Semantic Wikis [48], that provide the possibility to enrich the wiki content by semantic annotations, thus formulating explicit knowledge. The annotations correspond to ontological concepts, and knowledge reuse is improved by semantic search and semantic navigation. At the same time, Semantic Wikis successfully serve as ontology development tools, that offer a simple and web-based interface to build semantic applications. Current examples of Semantic Wiki implementations are, for instance, IkeWiki [47], KnowWE [7], MoKi [20], Semantic MediaWiki [30], and SweetWiki [14].

The knowledge in a Semantic Wiki is typically organized as follows: Every wiki article represents a concept of the ontology, and the content of the article informally describes the concept. Properties of the concept are defined by explicit semantic annotations within the article, where the annotations often link to other articles and concepts, respectively. In general, most Semantic Wiki systems are capable of developing and maintaining ontologies with the expressiveness of a subset of OWL [1]. Whereas this level of expressiveness is sufficient in many domains, some applications need the integration of knowledge beyond the power of

OWL. In our case, the development of (diagnostic) knowledge systems requires the representation of strong problem-solving knowledge, such as (production) rules, decision trees, or fault models. In this section, we introduce the Semantic Wiki KnowWE, that is extended by markups and interfaces to develop and share strong problem-solving knowledge. In consequence, KnowWE can be used as a web-based knowledge engineering tool for building (diagnostic) decision-support systems.

With the extension of a Semantic Wiki by strong problem-solving methods a number of implications arise:

- Concerning the internal layers of the system
  - Representation of the problem-solving knowledge in the semantic layer.
  - Processing the problem-solving knowledge by extended inference methods.
- Concerning the user interface
  - Interfaces for the acquisition and evolution of problem-solving knowledge.
  - Appropriate interfaces for using the knowledge.

In the following, we discuss these implications in more detail, and we show how these issues are mapped to the implementation of the system KnowWE. To the knowledge of the authors, KnowWE is the first implementation of a Semantic Wiki that integrates strong problem-solving knowledge into the wiki context.

We first motivate the use of the wiki by a small example, and then we discuss the underlying architecture in more detail. Throughout the rest of the paper, we use a simplified diagnosis system for car faults as the running example.

2.1 KnowWE by Example

In this section, we introduce the basic features of KnowWE by using a simple example application for diagnosing car faults. The basic idea is, that possible causes of a car fault—the solutions of the problem—are represented by corresponding wiki articles. The wiki contains, for instance, articles about *flat battery*, *clogged air filter*, and *bad ignition timing*.

In Figure 1, a page of the wiki is shown, describing the solution *bad ignition timing*. Besides standard text describing the problem in more detail, also explicit problem-solving knowledge is included on the page. At Figure 1-(1), two heuristic rules [42] of the rule base are displayed, that describe derivation knowledge of the solution. The first rule states, that the solution *Bad ignition timing* will receive a negative score, if the user enters for the symptom *engine start* that it neither *does not start* nor *barely starts*. A negative score decreases the evaluation score of the solution in the given case. The second rule states the derivation of the solution with respect to observations regarding the engine noises: The solution will receive a positive score, if the *engine noise* was observed by the user as *ringing* or *knocking*. In total, the example rule base for the solution *Bad ignition timing* comprises 11 rules. Besides the representation of rules, we also allow for the inclusion of model-based knowledge and decision trees; cf. Section 3 for more details.

We see that the derivation knowledge of a solution is locally defined and maintained together with the corresponding article of the solution; see Figure 9 for an example, where a rule base is edited in the article. This allows for a simplified update of informal (e.g., text) *and* explicit knowledge (e.g., rules) about one entity.

Although the wiki is mainly used as a tool for knowledge engineering, it also provides interfaces for interactive problem-solving. We give an example of the problem-solving process in the following: Some parts of the text are related to concepts of the knowledge base,

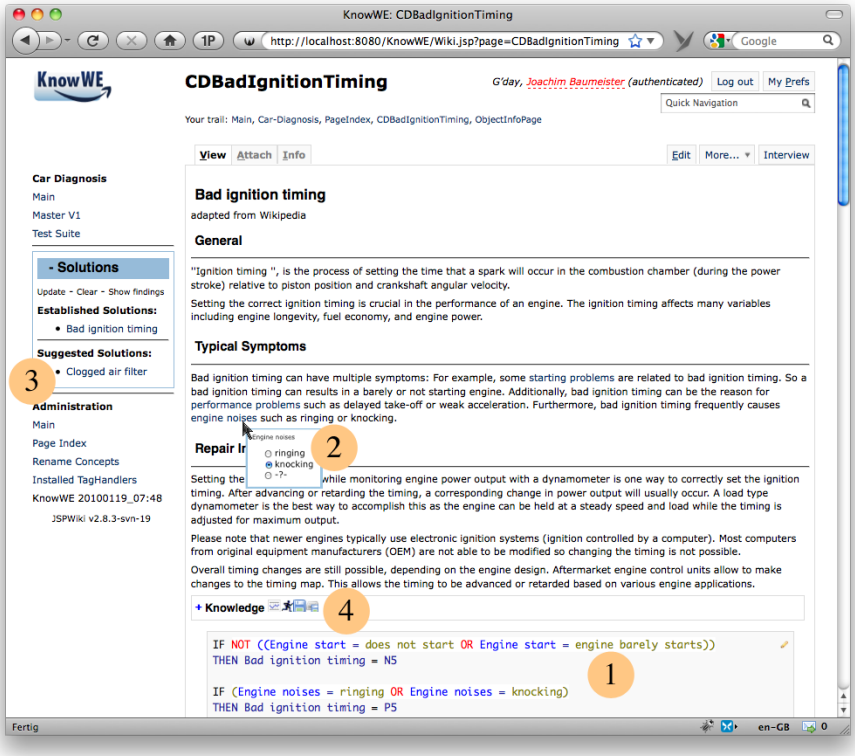

Fig. 1 A wiki article describing the solution *bad ignition timing* in the context of a car diagnosis application.

and thus have a meaning for the problem-solving process. Specific semantic annotations relate these text parts with the concepts. In the view mode of the wiki the user is able to click on the annotated text and can enter findings based on the corresponding concept. We call this approach *inline answers* for problem-solving in wikis. In Figure 1-(2) the text phrase "engine noises" was annotated by the corresponding concept *Engine noises* available in the knowledge base. In the given example, the value *knocking* for the concept *Engine noises* was entered by the user. As we explain in the following sections, the distributed reasoning process of KnowWE enables, that all registered knowledge bases contained in the wiki are notified about this new finding, and suitable states of the solutions are derived. In the solutions pane of the wiki—see Figure 1-(3)—we see that the solution *Bad ignition timing* is derived with a high certainty, whereas the alternative solution *Clogged air filter* was also derived and is considered as a possible solution. Both solutions were derived on the basis of this finding and previously entered findings. By clicking on the solution *Clogged air filter* in the solutions pane, we quickly can navigate to the wiki article describing the corresponding article. In this example, we see that not only the knowledge of the current article is used for problem-solving, but all knowledge bases in the wiki contribute to this process.

Alternatively, the user is able to download an executable version of the knowledge base by clicking the download button, see Figure 1-(4). Then, the knowledge base of the article is provided as download in the d3web format. The system d3web is a freely available runtime engine written as open-source toolkit; see SourceForge [4] for more details. This way, the

knowledge bases can be developed using the wiki and can be exported later to an external application if required.

In the following sections, we describe the underlying processes of creating and using knowledge bases within the Semantic Wiki KnowWE. First, we show how parts of the wiki article are compiled into executable knowledge bases. Second, we discuss the representation of the knowledge in the ontology layer of the wiki, and we finally sketch the distributed reasoning process, that enables the derivation of solutions over the entire wiki.

2.2 Transformation of Wiki Articles to Knowledge Bases

Usually, Semantic Wikis formalize one ontology that is distributed over the wiki: Every concept is represented by one distinct wiki article and properties between the concepts are usually defined by semantic annotations within the wiki articles.

For the development of problem-solving knowledge we extend this approach by a slightly more distributed architecture. When saving the currently edited article, the content is saved as a standard wiki page. Semantic annotations—included in the text—are identified and the domain ontology is updated accordingly. Additionally, dedicated parsers process the problem-solving knowledge found in the text into an executable knowledge base. We discuss specific markups in more detail in Section 3.

With this workflow, every concept of the ontology is represented by a distinct wiki article. However, the problem-solving knowledge related to this concept is externalized to a compiled knowledge base. In Figure 2, the described workflow is depicted, showing that an article is stored in the wiki repository in multiple ways: First, the repository saves the raw

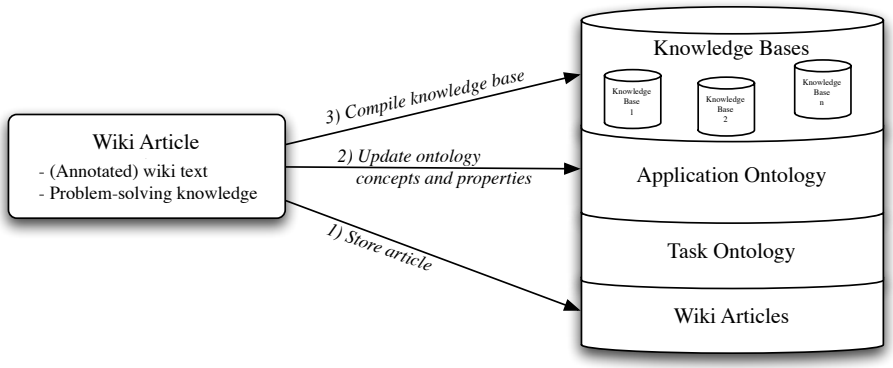

Fig. 2 After saving a wiki article, the knowledge is transformed and compiled into an executable format. The wiki repository holds the compiled knowledge bases together with the application ontology, the original wiki articles, and the general task ontology.

article in the *Wiki Articles* section of the repository (Fig. 2-1). Second, ontological concepts and properties, included in the article by semantic annotations, are stored/updated in the *Application Ontology* (Fig. 2-2). The compilation of the problem-solving knowledge contained in the article is filed to the *Knowledge Bases* section of the repository (Fig. 2-3). The foundational layer of all represented knowledge is the *Task Ontology*, that is used to connect

the described concepts and the problem-solving knowledge elements. The functioning of the task ontology is described in the next section.

We see, that the knowledge is redundantly stored as original text in the article repository, as ontological concepts in the application ontology repository, and as compiled knowledge base in the knowledge base repository. That way, we provide the knowledge in all formats that are required by the particular reasoners. For example, we use OWLIM [29] for OWL reasoning and the d3web engine [4] for processing the problem-solving knowledge. This redundant storage of the data and knowledge, respectively, helps to effectively use it for the later tasks.

2.3 A Task Ontology for Problem-Solving Knowledge

The foundation of all wiki articles is the *task ontology*: Here, the fundamental concepts of a Semantic Wiki integrating problem-solving knowledge are represented; examples are the relations between ontological concepts and text paragraphs in articles, but also principle concepts of the problem-solving task, such as *user input*, *solution*, and the connecting property *derives*.

2.3.1 Fundamental Concepts of the Task Ontology

The *task ontology* of KnowWE is the foundation of the system, since it represents the general entities of all applications built with the system. For example, it includes the definitions of findings and solutions, that are the basic elements of a problem-solving task, i.e., *findings* are used to *derive* particular *solutions*.

Figure 3 shows the most important concepts of the task ontology: As the key concept `Finding` holds a `Value`, that is assigned to an `Input`. Different inputs are structured into meaningful groups by the concept `Questionnaire`. Besides text and date inputs, the two main subclasses of `Input` are `Choice Input` and `Numeric Input`; they define attributes with discrete (named) values and numerical value ranges, respectively. An appropriate class of values for every input is defined by subclassing the concept `Value`, accordingly. The concept `Solution` denotes a special type of `Choice Input` in the hierarchy of inputs. The state of a solution is represented by a discrete finite value domain (`Solution Value`); its current value is not entered by the user but derived by problem-solving knowledge. The value range of a solution is restricted to one of the individuals `Established`, `Suggested`, `Undefined`, and `Excluded`. Further, the appropriate subclasses of `Value` are restricted to the corresponding `Input` subclasses; for instance, a `Solution Value` is restricted to be assigned only to instances of `Solution` concepts. Similarly, `Numeric Values` are assigned to `Numeric Inputs`. These and further property restrictions were omitted in Figure 3 in order to obtain a better overview of the basic concepts. We discuss these restrictions in more detail in the following.

2.3.2 Interweaving the Task Ontology and the Application Ontology

For a new knowledge engineering project the domain specialist defines the basic concepts (findings and solutions), that are relevant for the application domain. Ontological knowledge can be defined by semantic annotations included in the wiki article. When saving an article, the included concepts (together with corresponding properties) are stored in the application

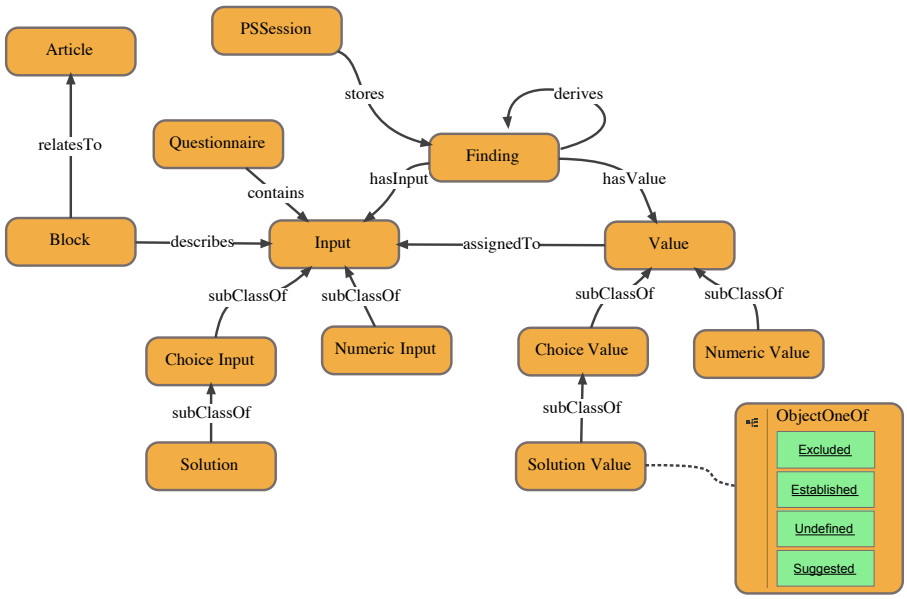

Fig. 3 Task ontology: Integrating problem-solving knowledge into a Semantic Wiki. Concepts of the task ontology are depicted in rounded rectangles, whereas instances are given by non-rounded rectangles (green).

ontology or—when already existing—are updated accordingly. For instance, input concepts of the application ontology are automatically aligned as subclasses of `Input` and `Value`, respectively. Analogously, solutions of the domain are described as subclasses of `Solution`. Figure 4 shows the task ontology (in orange rounded rectangles) extended by parts of an application ontology for the car diagnosis domain (in blue rectangles). We see that the concepts `Clogged air filter` and `Flat battery` are added as children to the concept `Technical problem`, which itself is a subclass of `Solution`. Also, the concept `Exhaust fumes` is defined as a choice input together with a corresponding concept `Exhaust fumes values`, that represent its possible value domain.

As we describe in Section 3, we provide explicit markups to define concepts of the application ontology. Thus, we are able to automatically align application concepts, specified in an article, as subclasses of an concept of the task ontology. For example, the concept `Exhaust fumes` is automatically aligned as a subclass of `Choice Input`. The interweaving of the task ontology with the application ontology is shown in Figure 5 in more detail. As described before, the property `assignedTo` between `Value` and `Input` is restricted in subclasses of both concepts. Thus, for instances of `Choice Value` only instances of `Choice Input` can be connected by the property `assignedTo`. During the construction of the application ontology, these restrictions are driven even further: For example, an instance of the concept `Exhaust fumes values` is limited to be assigned only to instances of the concept `Exhaust fumes`. In this manner, the particular values can be only assigned to the appropriate inputs. Further, we limit the alternatives of possible values for `Choice Inputs` by using a closed class for the corresponding `Value` concept: Only `black`, `blue`, and `invisible` are allowed as instances of the concept `Exhaust fumes values`. The use of universal quantifi-

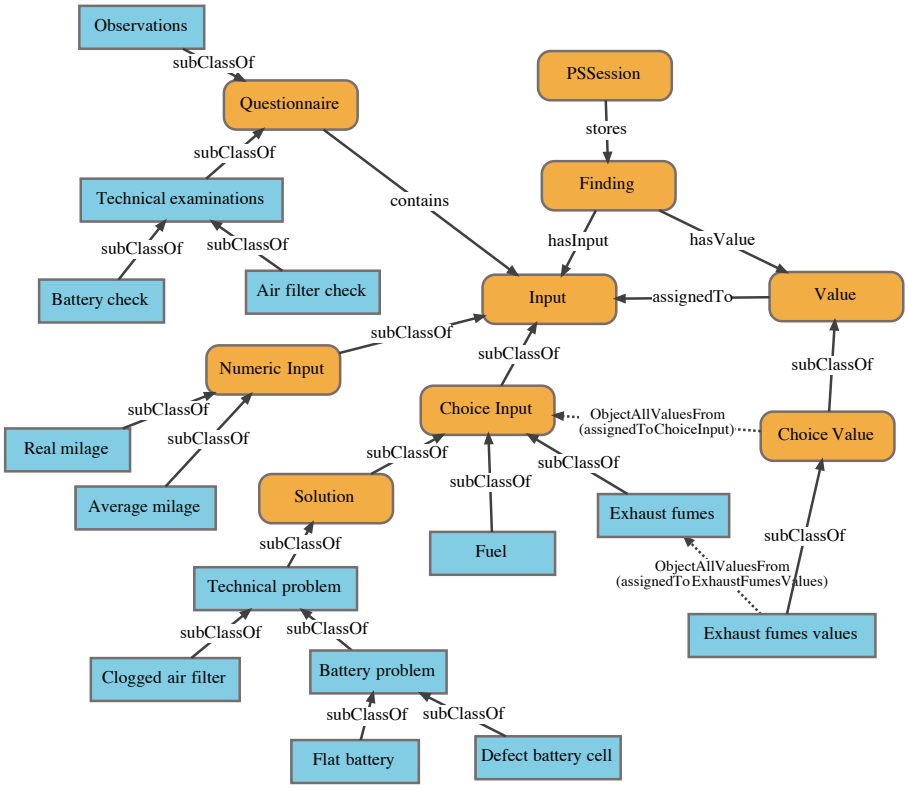

Fig. 4 Connecting the application ontology with the task ontology by subclassing. Concepts of the task ontology are depicted in rounded rectangles (orange), whereas concepts of the application ontology are represented by non-rounded rectangles (blue).

cation and closed classes guarantees only reasonable instances of a `Finding` concept during a problem-solving session.[1] It is important to notice, that the described ontological assertions between the task ontology and the application ontology are created/updated by the parsers when processing a modified wiki article.

2.3.3 Problem-Solving Sessions

The wiki allows for concurrent problem-solving by many users. Then, a distinct instance of `PSSession` is assigned to each user. The structure of the problem-solving process is depicted in Figure 6 by an example: A concrete problem-solving session (here with the name `user1`) is represented by an instance of the concept `PSSession`. Facts, entered by the user, are mapped to created instances of `Finding` (`Value` instances assigned to `Input` instances); derived solutions are represented as instances of solution values assigned to a specific instance of `Solution`. Currently, the system mostly utilizes external reasoners (rule engines, model-based systems, etc.) to derive solutions; due to a broker the state of externally derived

[1] There exist the corresponding axioms in OWL for closed classes and universal quantification `ObjectOneOf` and `ObjectAllValuesFrom`, respectively.

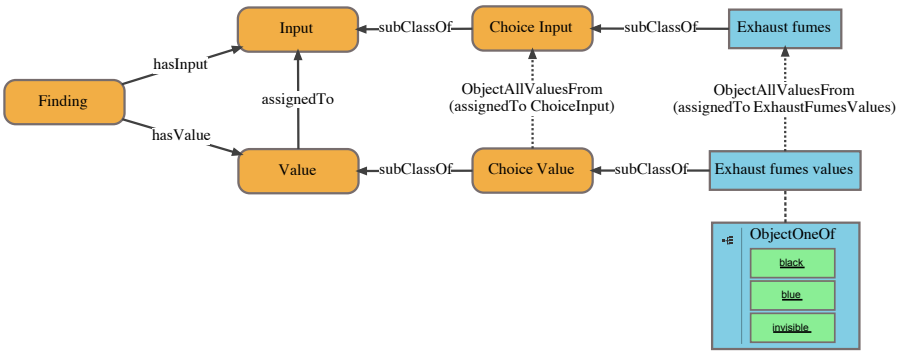

Fig. 5 Example: Subclassing the concept *Exhaust fumes* in the application ontology. Concepts of the task ontology are depicted in rounded rectangles (orange), concepts of the application ontology are given in non-rounded rectangles (blue), and instances are represented by green rectangles.

solutions is synchronized with the `Solution` instances of the corresponding `PSSession` instance (see Section 2.4 for more details). The reasoning processes of different users are inde-

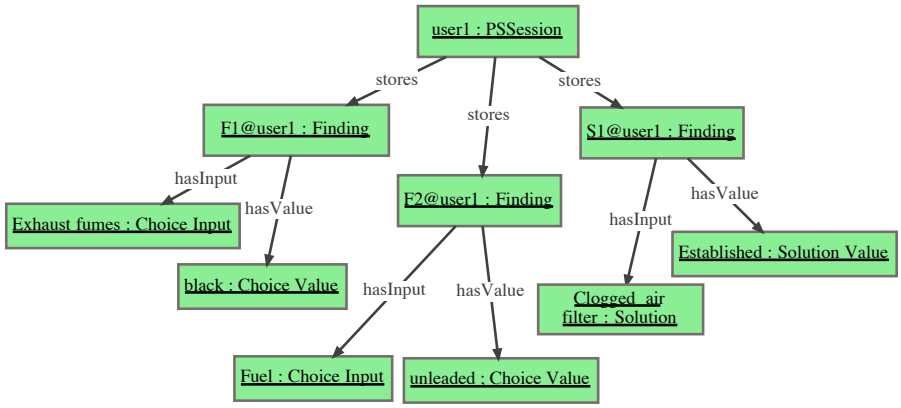

Fig. 6 Example: A problem-solving session instantiating two findings entered by the user (*Exhaust fumes=black* and *Fuel=unleaded*) and instance representing a derived solution (*Clogged air filter*).

pendent from each other, since every user is represented by a distinct instance of `PSSession`. In Figure 6 an example is given for the session instance `user1`: The problem-solving session instantiated two findings entered by the user (`Exhaust fumes=black` and `Fuel=unleaded`). Also, one solution was derived, that is represented by the instance `Clogged air filter = Established`. It is worth noticing, that for each fact a new instance only for the concept `Finding` is created; the actual input and value concepts are statically used as singeltons.

Related Approaches The use of different layers for task-specific and domain-specific knowledge is commonly suggested in the knowledge engineering research. Studer et al. [52] give an overview of principles and methods, that follow this distinction, for instance the

Expertise Model of CommonKADS [50] or the ontology layers of Protégé-II [16]. More recently, Crubezy and Musen [15] propose a similar approach: Here a *method ontology* specifies the required inputs/outputs of the problem-solving method and a *domain ontology* holds the application-specific knowledge. When compared to our approach, their use of the method ontology and domain ontology can be mapped to the task ontology and application ontology, respectively. Their framework, however, provides more flexibility due to the use of a separate *mapping ontology*. Whereas our approach automatically connects the concepts of the task ontology with the application ontology, these links are defined explicitly by a mapping ontology in their approach. In consequence, our approach requires less efforts due to the automatic alignment of the concepts, but limits the knowledge acquisition to diagnostic reasoning.

In the following section, we describe how new facts are derived by problem-solving knowledge, and how new derivations are mapped to new instances of `Finding`.

2.4 Distributed Inference

The distributed inference system of the wiki processes facts between the user and the available knowledge bases. New facts are added to the application ontology either in the form of findings entered by the user or due to solutions that are derived by knowledge bases. Every time, a new finding is entered by the user, the entry is mapped to a newly created instance of `Finding`. The new finding instance is stored in the application ontology, and a propagation of this new fact is started through the alignment service of the wiki. The alignment service maps the facts to corresponding entities, that are included in the knowledge bases. For example, the instance `Exhaust fumes = black` is mapped to the corresponding knowledge base object `Exhaust fumes`, that has the possible value `black`.

Based on this alignment the new fact is propagated to all registered knowledge bases. Some knowledge bases are able to use this fact to derive new facts, that are again propagated to the broker for further distribution. In Figure 7 this distributed reasoning process is depicted.

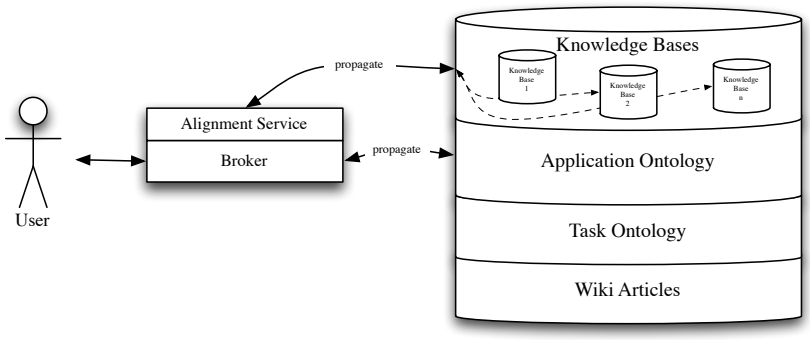

Fig. 7 Blackboard architecture of the distributed problem-solving within KnowWE.

That way, entered findings are not only processed by the knowledge base of the currently open wiki article, but also by all existing knowledge bases of the wiki. Therefore, all

solutions—that are represented in the wiki—can be derived anytime as it was motivated in Section 2.1.

Although this reasoning process is simple, its effectiveness depends on the quality of the alignment service. Currently, we implement a simple alignment of ontology concepts based on their naming and structure, i.e., concepts of the knowledge bases with the same name and the same value range are mapped to each other as equivalent classes. In the literature, however, more powerful approaches are described [17]. These can be added easily to the KnowWE system when required. Distributed reasoning offers a number of benefits when compared to a traditional monolithic problem-solving process: Due to the use of a collection of individual knowledge bases, the addition, modification, and exchange of single knowledge bases becomes easier. In principle, this architecture also allows for a spatially distributed reasoning process, i.e., the knowledge bases are situated on different, locally distributed servers.

However, in the context of knowledge engineering for decision-support systems it is often reasonable to control the problem-solving behavior of the edited knowledge bases. Thus, KnowWE provides the possibility to define a specific article as the *master* of the wiki: Here, a large coherent knowledge base is defined on the basis of imports of knowledge bases contained in other wiki articles. This master can be separately tested and exported as a single knowledge base. Further, we are able to define *variants* of knowledge bases by defining different masters importing varying collections of wiki articles. In Figure 8, an example

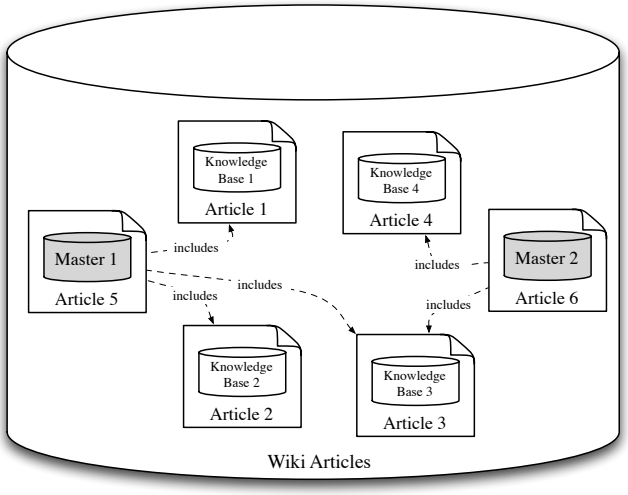

Fig. 8 Definition of knowledge variants by the specification of *masters* of the wiki.

of two masters of the wiki is depicted. In total, the wiki contains four wiki articles with knowledge bases, i.e., *Article 1* to *Article 4*. Additionally, the wiki page *Article 5* defines the master knowledge base *Master 1* by including the knowledge bases from the articles 1, 2, and 3. The alternative master *Master 2* defined in page *Article 6* only includes the knowledge bases from articles 3 and 4, and thus represents a different view of the entire knowledge base.

3 Knowledge Acquisition with Textual Markups

The knowledge acquisition interface always strongly depends on the problem-solving engine and knowledge representation used. KnowWE provides multiple markups to define problem-solving knowledge inline with the text, a mapping of the entered knowledge to the application ontology, and an integration of the reasoning results into the ontology and wiki interface (e.g., for showing derived solutions to the user). We integrated the open-source reasoning engine d3web [4] into the wiki, since it implements multiple reasoners for diagnostic inference and allows for a flexible adaptation of the knowledge engineering process with respect to the project requirements. As described in Section 2.3, the reasoning engine takes `Finding` instances from the application ontology and writes derived solutions/findings into the ontology again.

Due to the extensible architecture of KnowWE, it is possible to add further engines besides d3web, such as Prolog or JESS [19]; the architecture of the wiki was described in more detail in Reutelshoefer et al. [46].

In the following, we introduce markups to define terminological concepts, rules, decision trees, and set-covering models. The syntax of the particular markups should be as simple as possible in order to allow for an intuitive creation and evolution of the knowledge together with the standard wiki text. In the best case, motivated wiki users are capable of understanding and using the markup with only little training. With this ability, we enable an *ad-hoc knowledge engineering* process [45].

For an effective use, we propose to seamlessly integrate the acquisition of problem-solving knowledge into the standard authoring process of the wiki. For this reason, we embed the knowledge formalization into the edit pane of the wiki article, i.e., specialized textual markup is used to enter explicit knowledge inline with the wiki text. Figure 9 shows an example, where a rule base is included in the wiki article: Besides standard wiki markup for defining the content of text and images (see Figure 9-(1)), also a rule base for deriving the solution `Clogged air filter` is included, see Figure 9-(2).

It is possible to structure the wiki by solutions, i.e., each solution or group of related solutions defines a distinct wiki article. Besides standard wiki text and multimedia concerning the solutions, we also recommend including the corresponding problem-solving knowledge in the articles. The definition of the terminology is an exceptional case; here, we propose to define the possible inputs and their structure on a page, that is different from the pages describing its solution's derivation knowledge. That way, every article of a new solution can reuse the terminology for describing the problem-solving knowledge. Additional inputs—proprietary for a specific solution—can be either added ad-hoc on the solution's page or (after consulting the wiki admin) added to the distinct terminology page. Analogously, the taxonomy of solutions can be defined on a special page. Otherwise, every solution—not connected into a taxonomy—is added as direct subclass of the concept `Solution`, as shown in Figure 4.

3.1 Terminology Definition

In the previous section, we introduced the concept of the application ontology, that describes the basic entities of the considered domain. As for all other types of knowledge, the application ontology is defined within the edit pane of the wiki. We provide two hierarchies to define the elementary facts of the diagnostic task: (User) inputs and solutions. Both hierar-

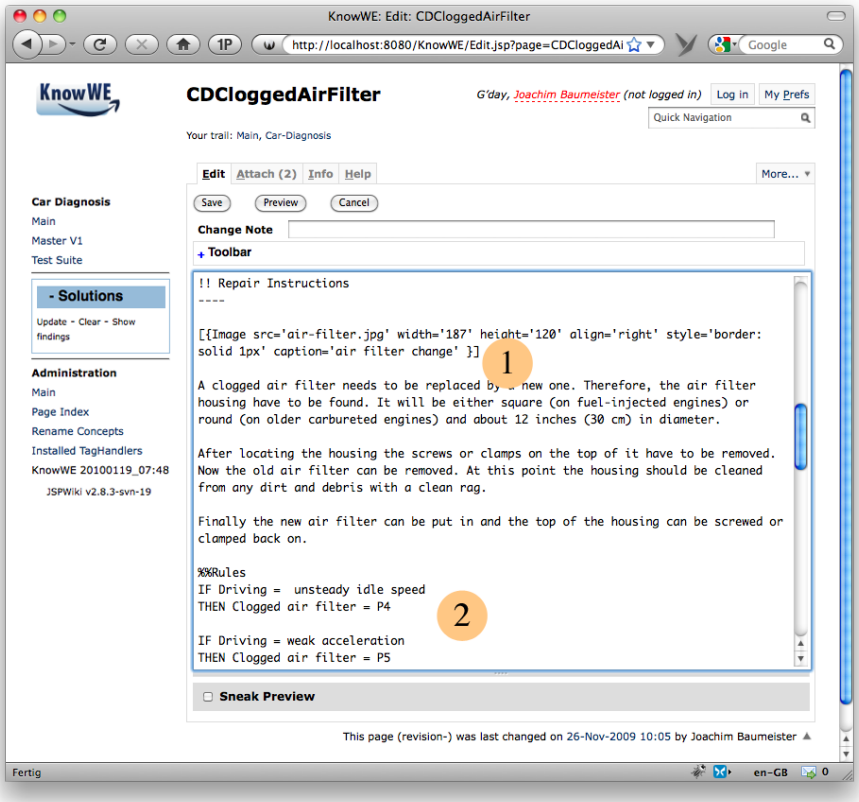

Fig. 9 Inline knowledge markup in KnowWE: (1) Inclusion of multimedia and (2) definition of derivation rules for the solution `Clogged air filter`.

chies are defined by so-called *dash-trees*. A dash-tree is a textual notation of a tree, where the successors of a node are represented by indenting dashes. This representation is quite common to textually visualize a tree-like structure.

```
Observations                          Technical problem
- Fuel [oc]                           - Clogged air filter
-- unleaded                           - Battery problem
-- diesel                             -- Flat battery
- Exhaust fumes [oc]                  -- Defect battery cell
-- black
-- blue
-- invisible
- Current fuel consumption [num]
- Average fuel consumption [num]
```

Above, two hierarchies are depicted as dash trees. In fact, we see that the dash-trees give an excerpt of the application ontology already shown in Figure 4. The left side shows an

input-tree, where inputs along with their values are defined. As required in the task ontology, inputs are grouped into questionnaires, and thus the root of every input tree denotes the name of the questionnaire the inputs are referring to. In the example, `Observations` is defined as a questionnaire containing the inputs `Fuel`, `Exhaust fumes`, `Current fuel consumption`, and `Average fuel consumption`. The first input `Fuel` is defined with the values `unleaded` and `diesel`. The marker `[oc]` denotes, that this input is defined as a `Choice Input`, i.e., an input with a discrete value range, where only one value can hold at a time. Analogously, the inputs `Current fuel consumption` and `Average fuel consumption` are defined as `Numerical Input` by the marker `[num]`.

The right side of the markup example shows the definition of the solution taxonomy. With `Technical problem` as the root solution, we define *is-a* relations to other solutions by dashes. For instance, the solutions `Clogged air filter` and `Battery problem` are sub-classes of `Technical problem`, whereas `Battery problem` has the two subclasses `Flat battery` and `Defect battery cell`. Since `Technical problem` is not connected as the child of another solution, it is automatically sub-classed by the concept `Solution` by default, see Figure 4. We see, that the dashes have different semantics for the input trees and the solutions trees; they represent the most common organization property for each hierarchy (contains vs. subClassOf).

The terminology of inputs and solutions can be defined anywhere in the wiki article, but both trees have to be wrapped by the tag %%`Questions ...`% and %%`Solutions ...`%, respectively, so that the system can identify the explicit terminology definitions. In the following sections, we introduce markups to define problem-solving knowledge, that is used to derive solutions based on given values of inputs.

3.2 Semantic Annotations

In KnowWE, semantic annotations are defined inline with the wiki text. The markup for those annotations was inspired by the syntax of Semantic MediaWiki [30], and ontological concepts can be simply linked by the definition of ontological properties. The general syntax of the markup connects a text phrase of the wiki text with a concept using an ontological property.

```
[Bad ignition timing is a technical problem
 <=> subClassOf:: TechnicalProblem] that can be solved ...
```

In the example shown above, the text phrase "Bad ignition ... problem" is annotated, stating, that the concept represented by this article is a `subClassOf` the concept `TechnicalProblem` — the relation is also shown in Figure 4. The annotation itself states, that the annotated text phrase documents/justifies the given relation.

By this type of annotation many useful ontological relations can be defined inline the wiki text. All annotations are represented in the application ontology and can be queried using a SPARQL [57] endpoint. SPARQL queries are embedded into the wiki text, and the results of the queries are shown in the view mode of the article.

Moreover, inline annotations also are used to define user inputs facilities. When using the `asks` annotation, the wiki renders access points to enter facts at the defined place. For example, the following annotation shows an excerpt of the edit pane corresponding to the wiki article shown in Figure 1.

```
!! Typical Symptoms

Bad ignition timing can have multiple symptoms: For example,
   ...
or weak acceleration. Furthermore, bad ignition timing
frequently causes [engine noises <=> asks:: Engine noises],
such as ringing or knocking.
```

In the second-last line, we see that the text phrase "engine noises" is annotated by the property `asks` with the concept `Engine noises` as value. This annotation yields the pop-up menu shown in Figure 1, where the user can enter a new finding instance related to the concept `Engine noises`.

3.3 Rules

KnowWE provides a specialized markup for the definition of rule-based knowledge. Rules are certainly the most popular knowledge representation for building knowledge bases. A rule

$$r = r.c \Rightarrow r.a$$

derives facts as defined in its consequent (rule action) *r.a*, if the specified rule condition *r.c* is satisfied. Facts derived by the rule can be interpreted as solutions, or as further inputs, that are used in conditions of other rules. The rule condition *r.c* typically consists of a combination of conjunctions and/or disjunctions constraining the values of inputs. For the sake of simplicity, we distinguish two basic types of rules, that can be used in the wiki:

1. *abstraction rules* for deriving new instances of findings, and
2. *scoring rules* for deriving a new state of a solution instance.

Abstraction rules simply define an input in their rule action, that is assigned by either a pre-defined value (for choice inputs) or is assigned by a numeric value. The numeric value can be either a static real value or can be computed by a formula given in the rule action.

In scoring rules we use *scores* to qualitatively derive solutions with a symbolic confirmation weight. These weights state the degree of confirmation or disconfirmation of a particular solution. Thus, a symbolic weight expresses the strength, for which the satisfied rule condition will confirm/disconfirm the solution. The definition and semantics of scoring rules goes back to the INTERNIST/QMR project [34] and the D3 system [41]. Analogous to the representation of the d3web rule system [2] we distinguish seven positive weights (P1, ..., P7) and seven negative weights (N1, ..., N7). Here, the weight P7 stands for the categoric derivation of a solution, the counter–weight N7 yields the categoric exclusion of a solution. The remaining weights are defined in a way, so that the aggregation of two equal weights results in the weight of the next higher weight, e.g., N3 + N3 = N4; two equal weights with opposite sign nullify, e.g., N3 + P3 = 0.

Due to the textual acquisition within wikis, the readability and intuitiveness of the markup is crucial for the effectivity of its application. Therefore, we decided to *not* use an already existing standardized markup for (horn clause) rules like RIF [27,56] or the languages SWRL/RuleML [25], but to promote a more compact and human–readable notation. In the following example two rules in the proposed markup are given: The abstraction rule r1 derives the fuel consumption with respect to the usual fuel consumption in percent. Here,

the value is computed by a formula including the values for `Average fuel consumption` and `Current fuel consumption`. The *scoring rule* r2 adds the weight P4 to the score of the solution `Clogged air filter`, if the user instance of `Fuel Evaluation` exceeds the value 140.

```
// Abstraction rule r1:
// Fuel consumption in percent to usual consumption
IF   Average fuel consumption > 0 AND
     Current fuel consumption > 0
THEN Fuel Evaluation = (Current fuel consumption  /
                        Average fuel consumption) * 100.0

// Scoring rule r2
// For an increased fuel consumption we increase the
// possibility of a clogged air filter.
IF   Fuel Evaluation > 140
THEN Clogged air filter = P4
```

Rules can be defined anywhere in the wiki article, but every coherent group of rules has to be wrapped by `%%Rules ...%`, so that the system scans this section for rules.

3.4 Decision Trees

The representation of classification knowledge using decision trees is very popular in machine learning research [43], but is also widely used for the manual definition of decision-support knowledge. The markup for the decision tree is very similar to the markup of the input terminology, as introduced in Section 3.1. The paths of the decision tree are represented by dashes indenting the particular inputs and values. Moreover, a decision tree also defines successors of input values to express, that the specified inputs are asked in case the input value was entered into the system.

The following example shows a simplistic decision tree for the derivation of a clogged air filter. Internally, decision trees are represented by questionnaires, so `Check air filter` is the questionnaire corresponding to the decision tree. The first question of the decision tree is related to the input `Fuel`, where the next input `Exhaust fumes` is asked when the instance `Fuel = unleaded` is present. If the follow-up input `Exhaust pipe color` then is answered with the value `black`, the solution `Clogged air filter` is derived with the value `established`; the categorical derivation is specified by the `(!)` annotation. Alternatively, scores—like introduced for scoring rules—can also be used (Section 3.3). If these scoring weights are used instead of the categoric derivation, then we call the tree a *heuristic decision tree* [42].

It is important to notice, that one decision tree can call other decision trees in its leafs instead of—or in addition to—deriving a solution. For example, the decision tree `Check diesel problems` is called, if the instance `Fuel = diesel` is present. The activation of other decision trees allows for the modularization of larger knowledge bases.

Also, decision trees do not only define knowledge for the derivation of knowledge, but also specify an interview strategy for a dialog with the user: Possible values of an input are denoted in an extra line of the markup. If such a value is followed by a user input with an

indent increased by one, then this input is interpreted as a follow–up input to be presented, when the previous input was answered with the given value.

```
Check air filter
- Fuel [oc]
-- unleaded
--- Exhaust fumes [oc]
---- black
----- Exhaust pipe color [oc]
------ black
------- Clogged air filter (!)
------ grey
---- blue
---- invisible
-- diesel
--- Check diesel problems

Check diesel problems
- Age of your car (in years) [num]
-- > 10
--- ...
-- <= 10
--- ...
```

Decision trees with the shown size are very easy to understand and to maintain. For larger decision trees, however, it is recommended to partition the tree logic into a set of sub–trees and refer to these sub–trees from a main tree. For example, the two decision trees from above—`Check air filter` and `Check diesel problems`—are refining trees, that are called from a main tree. The main tree itself is responsible for locating the coarse problem field and calling appropriate sub-trees.

Decision tree knowledge can be defined on arbitrary places in the wiki article, but they have to be wrapped by the terminology markup `%%Questions ...%`.

Since trees can be easily formulated using XML, it would have been an obvious approach to also represent the decision tree by an XML structure. Although this would yield a reduced compilation effort due to the existence of well–established XML parsers, the resulting markup appears to be too verbose and complex for standard wiki users. In contrast, the proposed markup is by far more compact and less vulnerable to syntax errors made by the user when formulating the decision tree.

3.5 Set-Covering Models

Decision trees and rule-based knowledge—as introduced above—consider the deductive derivation of solutions, which is common for the definition of classification knowledge. In some domains, however, it is preferable to use an abductive approach to formalize the derivation knowledge. Set-covering models [44] are a prominent example of an abductive knowledge representation. Albeit their basic representation is fairly simple, such models can be incrementally refined in order to improve the expressiveness of the knowledge [10].

The definition of the knowledge is very compact: Solutions are described by a group of findings, that are typically observed when the solution is present. We call these findings the *expected findings* of a solution. During the problem-solving process the *best-matching solution* is derived, i.e., the solution that computes the largest intersection between its expected findings and the currently present findings.

For the use of set-covering models in wikis we propose the application of *set-covering lists*. For each solution a collection of findings is defined, that are typically observed/entered when the solution is appropriate; the findings are enclosed in braces. The following example shows a set-covering list for the solution `Clogged air filter`:

```
Clogged air filter {
  Exhaust pipe color = black
  Exhaust fumes = black AND Fuel = unleaded
  Driving = IN (unsteady idle speed, weak acceleration)
  Fuel Evaluation > 140
  NOT (Exhaust fumes = black) [--]
}
```

Typical user inputs are listed, that are expected to be observed for the solution `Clogged air filter`, e.g., for unleaded gasoline we expect a black exhaust fume and pipe. A special markup is used with `[- -]` in the last line of the list: This annotation states, that the solution is *excluded* from the derivation, if the given finding is observed, i.e., the color of exhaust fumes is not black.

The lists can be defined anywhere in the wiki article, but they have to be wrapped by the tag %%`SetCoveringList ...`%.

3.6 Design and Evolution of Textual Markups

In the previous sections, we briefly introduced the possibility to define the terminology, rules, decision trees, and set-covering knowledge. KnowWE provides extended syntax to refine these types of knowledge, but also offers further markups, e.g., for the definition of decision tables. We refer the interested reader to the literature [8,45] for a more detailed discussion of available markups. In general, new markups and corresponding problem-solving knowledge can be easily added due to the plug-in architecture of KnowWE.

The markups—as presented in this article—underwent a continuous evolution in the past, where the system was used in case studies with students building toy systems. We observed typical errors and misunderstandings of the (little trained) users when applying the markup for building their knowledge bases. According to the identified issues we gradually refined and simplified the markups to their current state.

4 Management and Evaluation of Knowledge Bases

Up to that point, we described the concept of wiki-based knowledge engineering and we introduced suitable markups to define strong problem-solving knowledge. For the application of realistic knowledge engineering projects the management and evaluation of knowledge are critical issues.

4.1 Management

In the experienced development projects the size of the team was usually very small, ranging from 2 to 8 persons. Thus, a strict management approach was not necessary. In general, many projects developing strong problem-solving knowledge do not grow to a size that is greater than 5 people, and dedicated process models may complicate the overall task. With a growing team size, however, the implementation of project management rules becomes important, as they are known in ontology engineering, e.g., DILIGENT [55], in open-source software engineering [36], and in general design [18].

Independently from the team size, the specification of user roles during the project is helpful. In recent projects, a typical four–fold classification of roles was natural: Domain specialists, knowledge engineers, wiki champions, and users. Of course, for some persons some of the roles overlap. Here, *domain specialists* create and maintain the knowledge base together with the *knowledge engineers*, that provide support regarding its formalization and structure. The overall structure and evolution of the system is monitored by a *wiki champion*. The champion bears responsibility for the definition of the master articles and their quality, the deletion of pages, and the management of wiki contributors (creation and exclusion of wiki users). The definition of access rights for the particular user roles is supported by the built-in rights management of standard wiki software. In KnowWE rights are granted for read and write on page level.

4.2 Evaluation

Suitable evaluation methods heavily depend on the applied knowledge representation and the degree of formalization. In general, the characteristics of knowledge—as presented here—are special as to the following aspects:

1. Knowledge is distributed as *fragments* over the wiki articles. Different fragments may be owned by different persons.
2. The formalization degree of knowledge can vary between the particular fragments.
3. Different *variants* of the knowledge base are assembled by the definition of different master articles (see Figure 8).

In such a setting, it is not reasonable to jointly evaluate the knowledge of the entire wiki, but to run separate evaluations on the respective masters defined in the wiki. For each master, we commonly expect the knowledge to be at a uniform formalization level—an assumption that we cannot make for other variants of the knowledge.

For the *validation* of the knowledge, we provide a plugin for empirical test runs, i.e., the execution of (sequential) test cases [3]. Figure 10 shows a screenshot of the empirical testing plugin of KnowWE, where 100 test cases of the car diagnosis domain have successfully passed with a precision/recall of 1. A generated visualization of the test results can be also downloaded and used for manual inspection (as show on the right top of Figure 10). Test cases can be defined within the wiki and are executed together with a predefined master knowledge base. The *verification* of knowledge checks for inconsistency and other types of anomalies. Currently, KnowWE does not offer a built-in verification of the specified masters, but masters can be exported as knowledge bases. These knowledge bases are then verified using the workbench KnowME [2], that offers a variety of anomaly testing tools and uses the same file format for knowledge bases. Certainly, the tight integration of verification methods into the wiki is planned for future work, where also distributed methods are implemented [5].

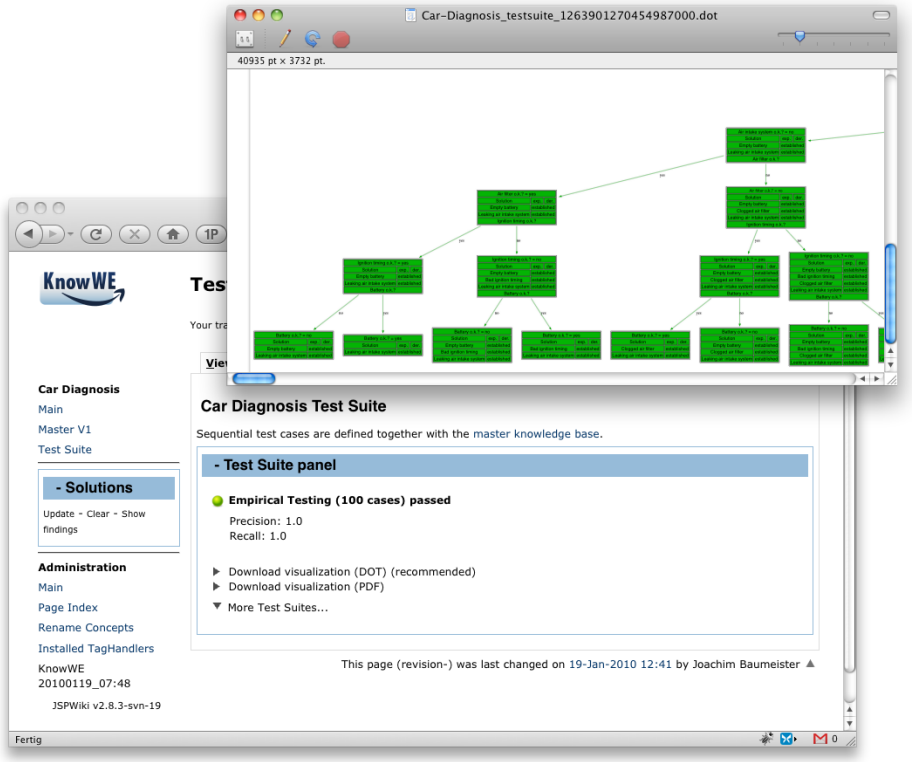

Fig. 10 Empirical testing with test cases in KnowWE; the results of the executed test cases can be visualized by DDTrees.

5 Case Study: The Digitalys CareMate System

The system is currently used in a number of (partly industrial, partly academic) projects, ranging from simple recommender systems to complex decision-support systems for technical and medical devices.

For example, KnowWE provides a technical platform to support a biological community within the *BIOLOG Wissen*[2] project (formerly LaDy). BIOLOG Wissen [37,6] serves as a web-based application for the collaborative construction and use of a decision-support system for landscape diversity. It aims to integrate knowledge on causal dependencies of stakeholders, relevant statistical data, and multimedia content. In another recent project, KnowWE is extended by diagnostic workflow knowledge in the context of the CliWE project[3]. By this extension, the wiki is used to collaboratively develop clinical guidelines, that are integrated as compiled knowledge bases into next-generation medical devices. A first prototype of this extension is reported in Hatko et al. [24].

In this paper, we describe the medical decision-support system *Digitalys CareMate*, that is currently maintained using the Semantic Wiki KnowWE.

[2] BIOLOG is funded by the German Federal Ministry of Education and Research from 2007-2009 (final funding phase).

[3] CliWE (Clinical Wiki Environment) is funded by Drägerwerk, Germany and runs from 2009-2011.

5.1 Medical Decision Support with CareMate

The decision support system *Digitalys CareMate* is commercially sold by the company Digitalys[4] as part of an equipment kit for medical rescue trucks. It is used as a consultation system during medical rescue missions, when the problem definition of a particular rescue service is complex and a second opinion becomes important.

The major goals of the project were the rated derivation of suitable solutions and the implementation of an efficient interview technique for busy rescue service staff in the emergency car. Thus, the user can be guided through an interview focussing on relevant questions of the current problem. With more questions answered the current ranking of possible solutions improves in relevance, and the interview strategy targets the presentation of reasonable follow-up questions. The interview strategy follows official school guidelines for emergency medical technicians in Germany.

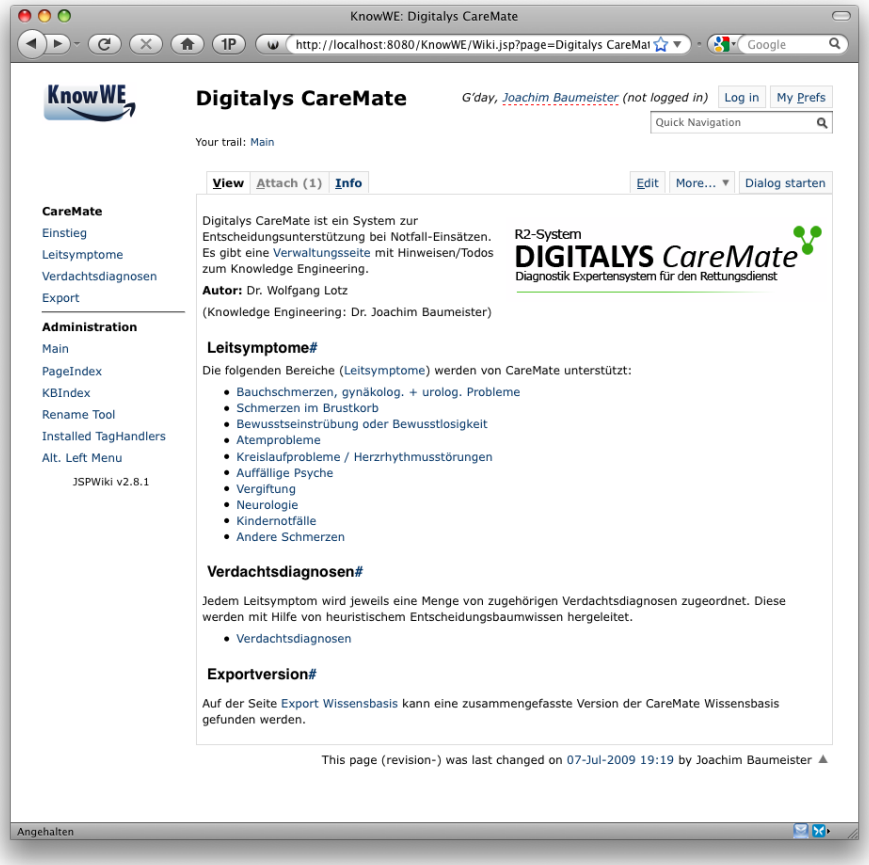

Fig. 11 Main screen of the Digitalys CareMate development environment – a KnowWE implementation (original screenshot is depicted in german language).

[4] http://www.digitalys.de

5.2 Structure and Engineering of the Knowledge Base

In the context of the CateMate project one of the most prominent assets of a (Semantic) Wiki was its freedom to structure the knowledge base. Thus, a wiki does not impose any restrictions on how to organize the knowledge base over the wiki articles. In this spirit, KnowWE also does not limit the developers with respect to the knowledge structure. For this reason, it became necessary to decide about the structuring of the CareMate knowledge at the beginning of the project. In the past, we experienced it to be natural at first to identify the core entities of the application domain; in a second step we try to structure the knowledge base according to instances of the identified entities.

For the CareMate project, the core entities are the *cardinal symptoms*, i.e., coarse findings describing vaguely the problem of the currently examined patient. The organization according to the cardinal symptoms is motivated by the observation, that in practice the emergency staff also tries to divide the problem by first identifying the cardinal symptom. Subsequently, the applicable domain knowledge can be easily partitioned with respect to the cardinal symptoms. The domain specialist provided the domain knowledge (interview strategy and solution derivation/rating) for each cardinal symptom in form of MS-Visio diagrams.

Each cardinal symptom is represented by a distinct wiki article, where—in the first step—the MS-Visio diagrams are uploaded as attachments and documentation/explanation is added as free text in the corresponding wiki articles. In the second step, the knowledge is formalized stepwise using the knowledge formalization pattern *heuristic decision tree* [42]; the markup for decision trees was introduced in Section 3.4. The use of heuristic decision trees was appropriate, since the dialog logic and the derivation behavior—as described in the MS-Visio diagrams—could be easily transcribed into a decision tree logic. An intermediate rating of solutions during the interview was possible, since heuristic decision trees allow for a scoring of solutions not only at the end of a dialog, but also in between the tree paths.

Figure 11 shows a screenshot of the main screen of the CareMate development environment. We see entry points for the 10 cardinal symptoms ("Leitsymptome"), for instance *neurological problems* ("Neurologie"), *chest pain* ("Schmerzen im Brustkorb"), and *disturbed consciousness* ("Bewusstseinstrübung und Bewusstlosigkeit"). Each cardinal symtom defines a decision tree that can be branched into subsequent decision trees. In their leafs, some trees link to other cardinal symptoms, when the interview progression shows, that another cardinal symptom is more relevant for the diagnostic process. A sophisticated interview strategy is implemented within the tree, so that an effective diagnosis can be made.

Thus, the wiki article of every cardinal symptom contains free text with documentation and explanations, the original knowledge source in form of MS-Visio diagrams, and the formalized knowledge fragment. For larger cardinal symptoms, there are further pages linked from the original article, where modular parts of the decision tree are formalized. In Figure 12 the wiki article of the cardinal symptom *stomach pain* ("Bauchschmerzen") is shown. Here, the wiki text describes that the decision tree logic was divided into two decision trees handling the diagnosis of stomach pain for women and for men, separately. For both decision trees an image is shown (can be enlarged on click), that gives an overview of the general structure of the questionnaire and the inference. The lower part of the browser window also shows an excerpt of the formalized knowledge base, where first the sex ("Geschlecht") of the patient is asked.

Whereas the semi-formalized specification of the workflows as MS-Visio diagrams took several weeks, the stepwise formalization as heuristic decision trees was done within some

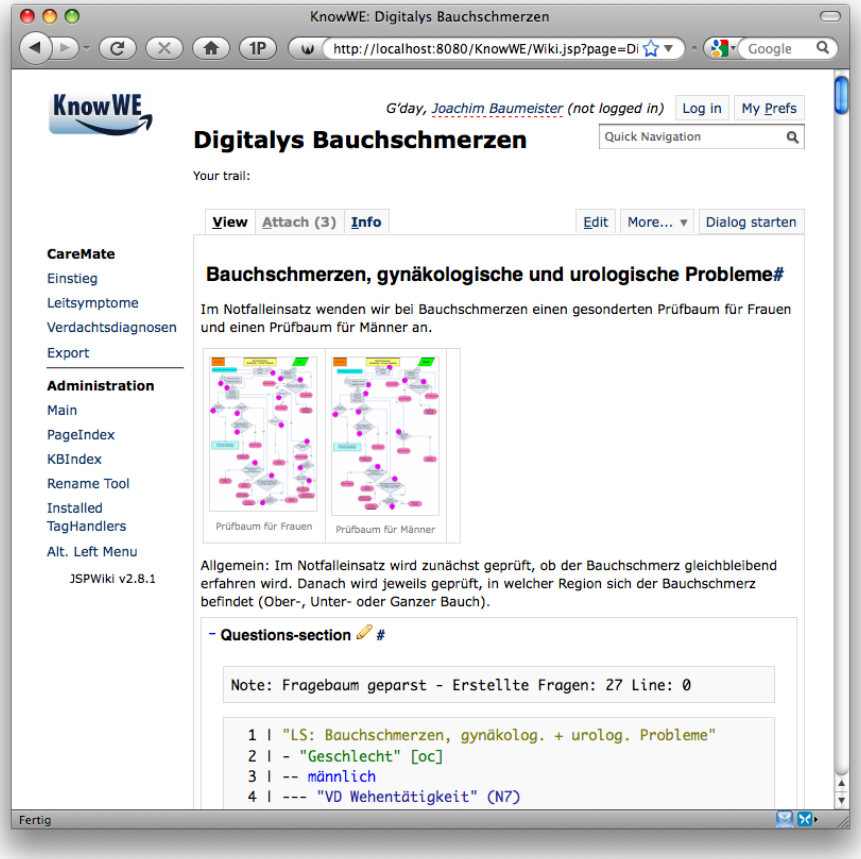

Fig. 12 Article of the cardinal symptom *stomach pain* ("Bauchschmerzen") showing some text explaining the general structure of the corresponding decision tree and overview images describing the knowledge in a graphical way. The images can be enlarged on click (original screenshot is depicted in german language).

days. The developed knowledge base was validated and verified using the DDTree visualization approach [3]. The wiki-based validation using test cases and its corresponding verification using the DDTree visualization is depicted in Figure 10.

At the current state of development, the knowledge base contains about 260 findings distributed over 33 questionnaires. There are about 145 distinct solutions represented by the system, where their derivation is logically organized by 10 cardinal symptoms. The compiled knowledge base resulted in a (merged) decision tree having 2051 possible diagnostic paths.

5.3 Export for a Runtime Version

Since, the knowledge base is running in a touch-screen application on a rough-sized notepad, releases of the knowledge base need to be exported from the wiki into the runtime. Therefore, we provide a deployment feature in KnowWE, where—on a separate wiki page—a

number of paragraphs of other wiki articles can be defined, that should be integrated into a compile process. More technically, we define paragraphs of wiki articles, that should be *included* into this new wiki article. By including the relevant parts of the knowledge definitions spread over the wiki, we are able to define a *view* of the entire knowledge base on this one wiki page. This join of knowledge bases was introduced as *masters* of the wiki previously in Section 2.4. The compiled knowledge base of this master article is exported for external use.

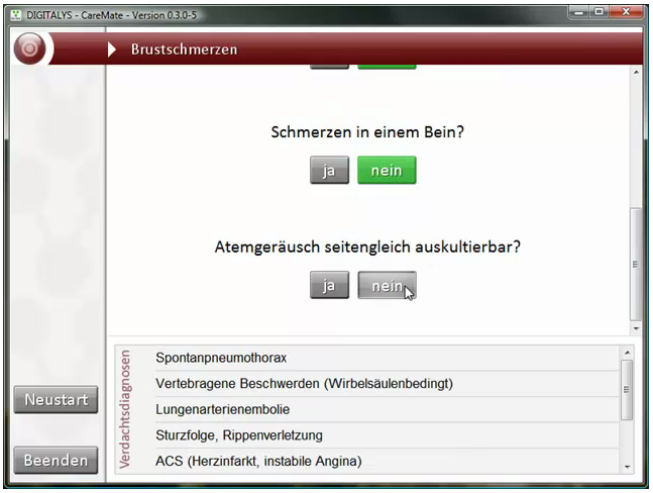

Fig. 13 Screenshot of the runtime application running the CareMate knowledge base. The center shows an auto-scrolling sequence of suitable inputs to be answered by the user. Current inputs are "Pain in a leg?" ("Schmerzen in einem Bein?") and "Bilateral respiratory sound?" ("Atemgeräusch seitengleich auskultierbar?"). The bottom pane shows an incrementally updated list of possible solutions, here with spontaneous pneumothorax ("Spontanpneumothorax") as the currently most probable solution. Due to the intended use of a touchscreen tablet the size of the buttons is enlarged (original screenshot depicted in german language).

Figure 13 shows a screenshot of the runtime application running the CareMate knowledge base. The center pane of the screen shows the currently active input to be answered by the user; previously answered inputs are automatically scrolled on top. In the bottom pane of the system, an ordered list of possible solutions is displayed. The list is updated incrementally when new findings are entered into the system. Since the application is installed on touchscreen tablet computers, the size of the buttons is increased appropriately.

5.4 Reflections on the Benefits of Using a Semantic Wiki

Semantic Wikis became prominent because of their support of community-driven knowledge engineering [49,31]. In a community-driven knowledge engineering approach, many people contribute to the distributed knowledge base. Whereas this approach fits especially for general domains, for instance *travel* [23], this may not always work in more specialized domains such as medical or biological science. Consequently, the presented project was traditionally developed by a single domain specialist as the principal knowledge contributor

and another specialist for reviewing the knowledge base. The more specialized a knowledge base is, the more advantages of a principal contributor exist. Often, the development of medical knowledge bases follows this approach, where a dedicated domain specialist or a small group of experts formalize their expertise in a particular domain.

Nevertheless, even for traditional development processes the use of Semantic Wikis supporting problem-solving knowledge is superior when compared to classic development environments. We sketch some of the advantages in the following:

1. **Flexible organization of the knowledge:** A Semantic Wiki provides *articles* as logical organization units. In a particular project the wiki makes no restrictions on how to fill this logical units and allows for any possible structure as long as it fits a partitioning into articles. In the presented project, we used the cardinal symptoms of the domain (*neurological problems*, *chest pain*, etc.) as the logical structure. This partitioning was reasonable with respect to the applied knowledge representation; we defined one or more heuristic decision trees for each cardinal symptom. In another project, the partitioning with respect to solutions can be more appropriate, i.e., defining an article for each solution, where the problem-solving knowledge for this solution is also embedded.
2. **Interweaving explicit and tacit knowledge:** In comparison to standard development tools for knowledge systems, a Semantic Wiki offers a simple combination of explicit knowledge, such as rules or decision trees, with tacit knowledge, for example text and images. Additional information represented in the wiki article can serve in many ways: 1) as *startup document* at the beginning of a project to informally collect knowledge about the domain, 2) as *documentation* of the knowledge engineering decisions taken, 3) as underlying *tacit knowledge* expressing the informal counter-part, and 4) as pursuing information for concepts represented by the article. In our project, the knowledge base was originally defined using MS-Visio documents. After the formalization, the documents were attached as underlying tacit knowledge explaining the decision tree representation in an alternative way. The files can be easily attached, and new versions of the documents do not overwrite older ones, but nicely integrate due to the automatic version control of the wiki. Thus, older versions can be reviewed and compared to the current state at any time. Moreover, the articles incorporated an informal documentation of the development process.
3. **Simple administration and rights management:** Many development tools require the installation of proprietary software on the client-side. With the use of a Semantic Wiki only a standard web-browser and an internet/intranet connection is necessary to start with the development process. Furthermore, this ability frees the knowledge engineers from the dependency of a particular computer; the development can be stopped and continued on any computer with a web-browser and an internet connection.
   Also, wikis provide a build-in rights management by default, that allows to restrict the read/write access of specific articles to a user or user group. Thus, parts of the wiki can be closed for the public, for example, because the knowledge engineering work is not finished there. Also, any content—knowledge or data—is held under version control, and thus changes and revisions can be safely performed.

6 Conclusions

Intelligent systems demonstrated their successful application in many domains. The costs of building and maintaining such systems, however, are still a critical problem. In this paper,

we identified two dilemmas, that often prevent successful knowledge engineering projects: The *single/multiple domain specialists* dilemma and the *flexibility/productivity* dilemma.

We claim that a flexible Semantic Wiki tailored to knowledge engineering tasks can help to relax these dilemmas. The paper introduced the Semantic Wiki KnowWE, which is an extended interpretation of standard Semantic Wikis by also providing the possibility to represent and use strong problem-solving knowledge for classification tasks. We showed, how classification knowledge is integrated into the semantic layer of the wiki, and we described the combined reasoning process of the ontology with the problem-solving knowledge. Integral components of KnowWE are the markups to represent variants of classification knowledge such as rules, decision trees, and set-covering models. The textual markups are embedded into the standard wiki text to formalize the problem-solving knowledge. We provided alternative formats and knowledge representations, respectively, to be able to flexibly adapt to possible project requirements. Technically, KnowWE was built on top of the wiki clone JSPWiki (http://www.jspwiki.org).

Besides classification problems, further classes of knowledge systems exist addressing, for instance, *configuration* and *scheduling*. When adapting the presented approach to tasks other than *classification*, we need to consider customizing the task ontology (Section 2.3) and the markups for the particular problem-solving knowledge (Section 3.3ff) together with an appropriate reasoning engine.

The system is used in some industrial and scientific projects; we demonstrated the application of the wiki by the engineering process of the Digitalys CareMate system — a medical decision-support system for emergency units. We discussed the use of a Semantic Wiki with respect to traditional development processes and we identified a number of advantages in comparison to classic development environments.

In the future, the evaluation and the evolution of knowledge in Semantic Wikis need to be considered more thoroughly. The *evaluation task* is not well-understood in the context of combining distributed problem-solving knowledge and ontologies. Both parts—expressive knowledge bases and ontologies—have been investigated in the past, but little research is available for a combined approach. Recently, the verification of ontologies with rules was investigated in Baumeister & Seipel [9], but no framework for the combination of general problem-solving knowledge with ontologies is known at the moment. Besides the verification of distributed knowledge, e.g., see Baumeister & Nalepa [5], also the validation of the knowledge needs to be considered in more detail. In addition, the evaluation of the explicit parts of the knowledge base in combination with the informal parts (text, multimedia) is also an open issue, that has not been solved sufficiently. Besides formal evaluation methods, the integration of socially-inspired evaluation approaches is an interesting issue, for example the application and the analysis of (advanced) user ratings [40].

Furthermore, experiences show that the maintainability of a knowledge base is critical for the longterm success of a system. In consequence, the *evolution* of distributed knowledge is an important research issue for the future, especially its evolution at different levels of formality. The current state–of–the–art provides separate works for ontology evolution [51, 38], and for the evolution of more expressive knowledge bases [21, 11]. Little work, however, has been done in the field of the combined evolution of knowledge at different levels of formality. Semantic Wikis provide the possibility to synchronously represent knowledge at different levels, for example in textual form and as a rule base. For the evolution of knowledge it appears natural, that also a combined approach for the modification of all existing knowledge elements can be provided.

Acknowledgements The described work was partly supported by the Digitalys GmbH in the context of the CareMate project; here, we especially would like to thank Dr. Wolfgang Lotz, Thomas Behra, and Timo Rumland. The authors would also like to thank—among others—Volker Belli, Martina Freiberg, Fabian Haupt, Reinhard Hatko, and Peter Klügl for their valuable contributions to the KnowWE development.

References

1. Grigoris Antoniou and Frank van Harmelen. Web ontology language: OWL. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*. Springer-Verlag, 2003.
2. Joachim Baumeister. *Agile Development of Diagnostic Knowledge Systems*. IOS Press, AKA, DISKI 284, 2004.
3. Joachim Baumeister. Advanced methods for empirical testing. In *FLAIRS'09: Proceedings of the 22th International Florida Artificial Intelligence Research Society Conference*, pages 378–383. AAAI Press, 2009.
4. Joachim Baumeister et al. The knowledge modeling environment d3web.KnowME. open-source at: http://d3web.sourceforge.net, 2008.
5. Joachim Baumeister and Grzegorz J. Nalepa. Verification of distributed knowledge in semantic knowledge wikis. In *FLAIRS'09: Proceedings of the 22th International Florida Artificial Intelligence Research Society Conference*, pages 384–389. AAAI Press, 2009.
6. Joachim Baumeister, Jochen Reutelshoefer, Fabian Haupt, and Karin Nadrowski. Capture and refactoring in knowledge wikis – coping with the knowledge soup. In *SCOOP'08: Proceedings of 2nd Workshop on Scientific Communities of Practice*, Bremen, Germany, 2008.
7. Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe. KnowWE – community–based knowledge capture with knowledge wikis. In *K-CAP '07: Proceedings of the 4th International Conference on Knowledge Capture*, pages 189–190, New York, NY, USA, 2007. ACM.
8. Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe. Markups for knowledge wikis. In *SAAKM'07: Proceedings of the Semantic Authoring, Annotation and Knowledge Markup Workshop*, pages 7–14, Whistler, Canada, 2007.
9. Joachim Baumeister and Dietmar Seipel. Anomalies in ontologies with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, in press, 2010.
10. Joachim Baumeister, Dietmar Seipel, and Frank Puppe. Incremental development of diagnostic set-covering models with therapy effects. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 11(Suppl. Issue 2):25–49, November 2003.
11. Joachim Baumeister, Dietmar Seipel, and Frank Puppe. Refactoring methods for knowledge bases. In *EKAW'04: Engineering Knowledge in the Age of the Semantic Web: 14th International Conference, LNAI 3257*, pages 157–171, Berlin, 2004. Springer.
12. Joachim Baumeister, Dietmar Seipel, and Frank Puppe. Agile development of rule systems. In Giurca, Gasevic, and Taveter, editors, *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*. IGI Publishing, 2009.
13. B.G. Buchanan and E.H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.
14. Michel Buffa, Fabien Gandon, Guillaume Ereteo, Peter Sander, and Catherine Faron. SweetWiki: A semantic wiki. *Web Semantics*, 8(1):84–97, 2008.
15. Monica Crubézy and Mark Musen. Ontologies in support of problem solving. In *Handbook on Ontologies in Information Systems*, pages 321–342. Springer, 2004.
16. Henrik Eriksson, Yuval Shahar, Samson W. Tu, Angel R. Puerta, and Mark Musen. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79:293–326, 1995.
17. Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, Berlin, 2007.
18. Gerhard Fischer. Social creativity: turning barriers into opportunities for collaborative design. In *PDC'04: Proceedings of the 8th Conference on Participatory Design*, pages 152–161. ACM Press, 2004.
19. Ernest Friedman-Hill. *Jess in Action: Java Rule-based Systems*. Manning, 2003.
20. 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*, pages 831–835. Springer, 2009.
21. Yolanda Gil and Marcelo Tallis. A script-based approach to modifying knowledge bases. In *AAAI/IAAI'97: Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference*, pages 377–383, 1997.

22. William Grosso, Henrik Eriksson, Ray W. Fergerson, John H. Gennari, Samson W. Tu, and Mark Musen. Knowledge Modeling at the Millennium – The Design and Evolution of Protégé-2000. In *Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW 1999)*, Banff, Canada, 1999.
23. Tom Gruber. Collective knowledge systems: Where the Social Web meets the Semantic Web. *Web Semantics*, 6(1):4–13, February 2008.
24. Reinhard Hatko, Volker Belli, and Joachim Baumeister. DiaFlux: Diagnostic flows in wikis. In *FGWM'09: Proceedings of German Workshop of Knowledge and Experience Management (at LWA'09)*, 2009.
25. Ian Horrocks, Peter F. Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov. OWL Rules: A Proposal and Prototype Implementation. *Web Semantics*, 3(1):23–40, 2005.
26. Matthias Hüttig, Georg Buscher, Thomas Menzel, Wolfgang Scheppach, Frank Puppe, and Hans-Peter Buscher. A Diagnostic Expert System for Structured Reports, Quality Assessment, and Training of Residents in Sonography. *Medizinische Klinik*, 3:117–22, 2004.
27. Michael Kifer. Rule Interchange Format: The framework. In Nick Bassiliades, Guido Governatori, and Adrian Paschke, editors, *RuleML'08: Rule Representation, Interchange and Reasoning on the Web*, volume 5321 of *Lecture Notes in Computer Science*, pages 1–2. Springer, 2008.
28. Mikio Kimura, Mitsuo Sakamoto, Takuya Adachi, and Hiroko Sagara. Diagnosis of febrile illnesses in returned travelers using the PC software GIDEON. *Travel Medicine and Infectious Disease*, 3(3):157–160, 2005.
29. Atanas Kiryakov. OWLIM: Balancing between scalable repository and light-weight reasoner. In *WWW'06: Proceedings of the 15th International Conference on World Wide Web*, 2006.
30. Markus Krötzsch, Denny Vrandecić, and Max Völkel. Semantic MediaWiki. In *ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273*, pages 935–942, Berlin, 2006. Springer.
31. Markus Krötzsch, Denny Vrandecić, Max Völkel, Heiko Haller, and Rudi Studer. Semantic Wikipedia. *Web Semantics*, 5(4):251–261, 2007.
32. William Lidwell, Kritina Holden, and Jill Butler. *Universal Principles of Design*. Rockport Publishers, October 2003.
33. Stefan Mersmann and Michel Dojat. SmartCare$^{\text{tm}}$ - automated clinical guidelines in critical care. In *ECAI'04/PAIS'04: Proceedings of the 16th European Conference on Artificial Intelligence, including Prestigious Applications of Intelligent Systems*, pages 745–749, Valencia, Spain, 2004. IOS Press.
34. Randolph A. Miller, Harry E. Pople, and J. Myers. INTERNIST-1, an Experimental Computer-Based Diagnostic Consultant for General Internal Medicine. *New England Journal of Medicine*, 307:468–476, 1982.
35. Robert Milne and Charlie Nicol. TIGER: Continuous diagnosis of gas turbines. In *Proceedings of the 14th European Conference on Artificial Intelligence*, Berlin, Germany, 2000.
36. Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions in Software Engineering Methodology*, 11(3):309–346, July 2002.
37. Karin Nadrowski, Joachim Baumeister, and Volkmar Wolters. LaDy: Knowledge Wiki zur kollaborativen und wissensbasierten Entscheidungshilfe zu Umweltveränderung und Biodiversität. *Naturschutz und Biologische Vielfalt*, 60:171–176, 2008.
38. Natalya F. Noy, Abhita Chugh, William Liu, and Mark A. Musen. A framework for ontology evolution in collaborative environments. In *ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273*, pages 544–558, 2006.
39. Natalya F. Noy, William Grosso, and Mark Musen. Knowledge-acquisition interfaces for domain experts: An empirical evaluation of Protégé-2000. In *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE 2000)*, Chicago, USA, 2000.
40. Natalya F. Noy, Ramanathan Guha, and Mark Musen. User ratings of ontologies: Who will rate the raters? In *Proceedings of the AAAI 2005 Spring Symposium on Knowledge Collection from Volunteer Contributors*, Stanford, CA, 2005. AAAI Press.
41. Frank Puppe. Knowledge Reuse among Diagnostic Problem-Solving Methods in the Shell-Kit D3. *International Journal of Human-Computer Studies*, 49:627–649, 1998.
42. Frank Puppe. Knowledge formalization patterns. In *PKAW 2000: Proceedings of the Pacific Rim Knowledge Acquisition Workshop*, Sydney, Australia, 2000.
43. J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.
44. James A. Reggia, Dana S. Nau, and Pearl Y. Wang. Diagnostic Expert Systems Based on a Set Covering Model. *Journal of Man-Machine Studies*, 19(5):437–460, 1983.
45. Jochen Reutelshoefer, Joachim Baumeister, and Frank Puppe. Ad-hoc knowledge engineering with semantic knowledge wikis. In *SemWiki'08: Proceedings of 3rd Semantic Wiki workshop - The Wiki Way of Semantics (CEUR Proceedings 360)*, 2008.

46. Jochen Reutelshoefer, Florian Lemmerich, Fabian Haupt, and Joachim Baumeister. An extensible semantic wiki architecture. In *SemWiki'09: Proceedings of the 4th Workshop on Semantic Wikis – The Semantic Wiki Web (CEUR proceedings 464)*, 2009.
47. Sebastian Schaffert. IkeWiki: A semantic wiki for collaborative knowledge management. In *STICA'06: 1st International Workshop on Semantic Technologies in Collaborative Applications*, Manchester, UK, 2006.
48. Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel. Semantic wiki. *IEEE Software*, 25(4):8–11, 2008.
49. Sebastian Schaffert, Andreas Gruber, and Rupert Westenthaler. A semantic wiki for collaborative knowledge formation. In *Proceedings of SEMANTICS 2005 Conference*. Trauner Verlag, 2006.
50. Guus Schreiber, Hans Akkermans, Anjo Anjewierden, Robert de Hoog, Nigel Shadbolt, Walter Van de Velde, and Bob Wielinga. *Knowledge Engineering and Management - The CommonKADS Methodology*. MIT Press, 2 edition, 2001.
51. Ljiljana Stojanovic, Alexander Maedche, Boris Motik, and Nenad Stojanovic. User-driven ontology evolution management. In *EKAW'02: Ontologies and the Semantic Web, 13th International Conference, LNAI 2473*, pages 285–300, Berlin, 2002. Springer.
52. Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.
53. York Sure, Michael Erdmann, Jürgen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. OntoEdit: Collaborative ontology development for the Semantic Web. In *ISWC'02: International Semantic Web Conference*, pages 221–235, 2002.
54. York Sure, Steffen Staab, and Rudi Studer. On-to-knowledge methodology (OTKM). In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies, International Handbooks on Information Systems*, pages 117–132. Springer, 2004.
55. Denny Vrandecić, H. Sofia Pinto, York Sure, and Christoph Tempich. The DILIGENT knowledge processes. *Journal of Knowledge Management*, 9(5):85–96, October 2005.
56. W3C. RIF-BLD Specification: http://www.w3.org/tr/rif-bld, July 2008.
57. W3C. SPARQL recommendation: http://www.w3.org/tr/rdf-sparql-query, January 2008.