Web–Based Knowledge Engineering with Knowledge Wikis

Joachim Baumeister and **Frank Puppe**
Institute of Computer Science, University of Würzburg, Würzburg, Germany
baumeister/puppe@informatik.uni-wuerzburg.de

Abstract

Knowledge engineering research have thoroughly considered the problem of distributed knowledge acquisition and management over the last decades. In this paper, we claim that the success of the Web 2.0 platforms in conjunction with semantic technologies will have an impact on distributed knowledge engineering. Especially, (extended) semantic wikis offer a well–accepted tool allowing for a simplified access to the development process as well as providing an intuitive interface for knowledge sharing. We introduce the concept of *knowledge wikis* that are used as a collaborative knowledge engineering and sharing environment. Here, knowledge can be captured and used at different levels of detail ranging from text in natural language to formal rule bases.

Motivation

The challenge of building intelligent systems has been tackled by knowledge engineering research over the last decades. With the appearance of the web and the semantic web vision, respectively, the world wide web has become the main infrastructure for sharing knowledge in general. Intelligent systems on the web are accessible to an arbitrary number of users, and also for knowledge acquisition interfaces on the web the knowledge engineering process can become distributed in a simple manner.

Prominent examples like Wikipedia have shown that standard users are willing to provide and maintain knowledge on the web. More precisely, the wiki technology has displayed its impact for creating and sharing knowledge not only in the open web environment, but also for a limited group of users, e.g., as knowledge management tools in companies. The most important aspect of a wiki is the ability to change and refine its content immediately: Any wiki page can be simply modified using a web browser by the mandatory edit feature of the wiki. Changes are then directly presented after saving the modifications.

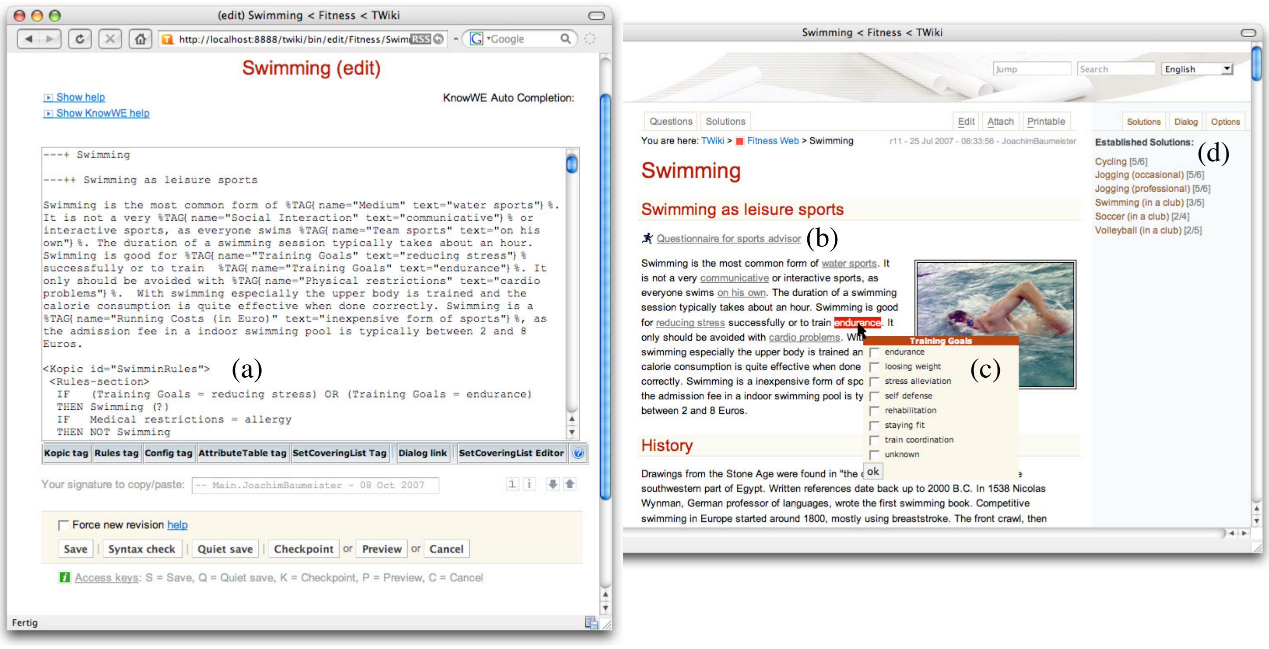
However, many existing wikis only allow for an unstructured and informal representation of knowledge that is consumable by humans but not by machines. Recently, semantic wikis, e.g., (Krötzsch, Vrandecić, & Völkel 2006) emerged to extend normal wikis by semantic annotations of the wiki content relating the textual content to a formal ontology. Whereas some of these wikis tend to be rather flat extensions of normal wikis, e.g., (Krötzsch *et al.* 2007), some implementations are targeted to be used as ontology engineering environments, e.g., (Schaffert 2006; Auer, Dietzold, & Riechert 2006).

An intriguing question is whether it is feasible to transfer the idea of (semantic) wikis to knowledge engineering, i.e., to investigate whether wikis can be used to create, share, and maintain explicit problem-solving knowledge in a comparably simple manner as they are able to create and share textual information/knowledge.

In a knowledge wiki the knowledge base is entered together with the standard text in the usual edit pane of the wiki using appropriate textual knowledge markups, e.g., as proposed in (Baumeister, Reutelshoefer, & Puppe 2007). The most important characteristics of these markups are the intuitive understanding and use even for less experienced users. Furthermore, the markup has to be compact in order to fit into the usual flavor of a wiki, i.e., compact and explicit but intuitively understandable. In summary, the markup should support the formation of *ad-hoc knowledge engineers*.

When saving the wiki page the included textual representation of the knowledge is extracted and compiled into an executable knowledge base corresponding to the particular wiki article. Solutions and user inputs in different knowledge bases but with identical names and identical structure are aligned automatically. More complex alignments can be defined by specific alignment rules. The compiled knowledge can either be used by starting an interactive interview with the user or by in–place answers that are provided by the user with the text of the knowledge wiki. Then, pop-up menus appear at the place of annotated text phrases that enable the user to answer questions during browsing the text. Any user input entered into the system – either by an interactive interview or by in–place answers – is propagated to the reasoning engine of the knowledge wiki. In consequence, possibly derived solutions are shown in the wiki.

In this paper, we introduce the concept of distributed knowledge engineering with knowledge wikis, and we demonstrate the methods and techniques using the toy application of a *sports advisor wiki*. The running example



```
---+ Swimming

---++ Swimming as leisure sports

Swimming is the most common form of %TAG{ name="Medium" text="water sports"} %.
It is not a very %TAG{ name="Social Interaction" text="communicative"} % or
interactive sports, as everyone swims %TAG{ name="Team sports" text="on his
own"} %. The duration of a swimming session typically takes about an hour.
Swimming is good for %TAG{ name="Training Goals" text="reducing stress"} %
successfully or to train  %TAG{ name="Training Goals" text="endurance"} %. It
only should be avoided with %TAG{ name="Physical restrictions" text="cardio
problems"} %.  With swimming especially the upper body is trained and the
calorie consumption is quite effective when done correctly. Swimming is a
%TAG{ name="Running Costs (in Euro)" text="inexpensive form of sports"} %, as
the admission fee in a indoor swimming pool is typically between 2 and 8
Euros.

<Kopic id="SwimminRules">
 <Rules-section>
  IF   (Training Goals = reducing stress) OR (Training Goals = endurance)
  THEN Swimming (?)
  IF   Medical restrictions = allergy
  THEN NOT Swimming
```

-- Main.JoachimBaumeister - 08 Oct 2007

Swimming

Swimming as leisure sports

Swimming is the most common form of water sports. It is not a very communicative or interactive sports, as everyone swims on his own. The duration of a swimming session typically takes about an hour. Swimming is good for reducing stress successfully or to train endurance. It only should be avoided with cardio problems. Wit swimming especially the upper body is trained an calorie consumption is quite effective when done correctly. Swimming is a inexpensive form of spo the admission fee in a indoor swimming pool is ty between 2 and 8 Euros.

- endurance
- loosing weight
- stress alleviation
- self defense
- rehabilitation
- staying fit
- train coordination
- unknown

History

Drawings from the Stone Age were found in "the c

southwestern part of Egypt. Written references date back up to 2000 B.C. In 1538 Nicolas Wynman, German professor of languages, wrote the first swimming book. Competitive swimming in Europe started around 1800, mostly using breaststroke. The front crawl, then

Solutions

Cycling [5/6]
Jogging (occasional) [5/6]
Jogging (professional) [5/6]
Swimming (in a club) [3/5]
Soccer (in a club) [2/4]
Volleyball (in a club) [2/5]

Figure 1: The knowledge wiki KNOWWE: (a) the edit interface showing textual descriptions and derivation rules for the solution *Swimming*, and (b–c) the corresponding view of the wiki page.

considers a wiki providing knowledge about different forms of sports, both in textual and in explicit manner. Explicit knowledge can be used to derive an appropriate form of sport for interactively entered (user) preferences. Besides such a simple recommendation application the wiki can be used for a variety of tasks briefly sketched in the case studies. The rest of the paper is organized as follows: In the following section we introduce the basic concepts of a knowledge wiki, i.e., aspects on knowledge acquisition, integration, and knowledge sharing. Reuse for building a knowledge wiki is improved when an upper ontology is used. Therefore, we introduce an upper ontology describing the relevant concepts needed for problem–solving (excluding the representation of the actual inference task), and we also introduce suitable markups for the specification of problem–solving knowledge. We discuss the experiences using a knowledge wiki in case studies and we conclude the paper with a summary and an outlook to future work.

The Knowledge Wiki KNOWWE

Usually, in semantic wikis every wiki page represents a distinct concept of the considered application domain. Knowledge wikis specialize this approach by representing a possible solution with every wiki page, i.e., a concept that can be derived in a problem–solving session. On every page the solution is described by textual information but also by multimedia content (e.g., figures or pictures). Furthermore, the solution is formally defined by explicit derivation knowledge, e.g., rules, that specify problem–solving knowledge inferring the particular solution. In our sports advisor example, for Swimming, Running, etc. a single wiki page is available containing unstructured information and explicit knowledge describing the form of sport. The implementation KNOWWE (Knowledge Wiki Environment) offers a variety of different knowledge representations and their use depend on the characteristics and complexity of the actual application domain.

When a user is looking for an appropriate solution, then he/she can either browse the contents of the wiki in a classic web–style but also is able to activate an interactive problem–solving interview asking for values of the represented inputs. Alternatively, the user can enter findings by clicking on in–place answers embedded in the normal wiki text. For the given findings suitable solutions are derived, globally, i.e., the knowledge of *all* solutions contained in the wiki is used in a distributed manner to infer solutions.

In general, a knowledge wiki extends the functionality of a semantic wiki by the representation, the reasoning, and the engineering of explicit problem–solving knowledge. In the following, we explain these aspects of a knowledge wiki in more detail.

Wiki-based Knowledge Acquisition

Like in semantic and normal wikis new content is entered by the mandatory edit interface of the system, which is typically available through a web browser. In addition, knowledge wikis allow for the definition of explicit knowledge to be used for problem–solving tasks.

Since all knowledge wiki applications follow a common understanding of how to represent the knowledge, we provide a *problem–solving ontology* serving as the upper ontology of all concepts defined in an application project. All defined concepts and properties are inherited by concepts and properties of this ontology in order to allow for a rapid development of new applications. Thus, new solutions are inherited from the standard concept *Solution*, whereas user inputs with their corresponding values are inherited from the standard concept *Finding*. The ontology is described in more detail in the following section.

Adding new Knowledge A new solution is added to the knowledge wiki by simply creating a new wiki page having the solution's name. By default, the new solution is added as a direct child of the standard concept *Solution*, but a more refined taxonomy of the entered solutions can be defined using a distinct wiki page. The wiki page includes describing text in natural language and the explicit knowledge for deriving the new solution.

For example, Figure 1(a) shows the edit pane of the knowledge wiki containing text describing the sports form *Swimming* together with rules for deriving the corresponding solution based on inputs such as *Training goals* and *Medical restrictions*. In consequence, the solution *Swimming* is added as a child of the concept *Solution*, if not specified otherwise, and the inputs *Training goals* and *Medical restrictions* and their corresponding findings (e.g., *Training goals = endurance*) are reused from a given application ontology.

Besides rules the implementation of KNOWWE offers different textual markups to define problem–solving knowledge like decision trees and set–covering models. We describe the markups for alternative knowledge representations in a separate section.

Alternatively, we offer the possibility to enter set–covering knowledge as inline annotations by tagging meaningful phrases of the wiki text. Distinguished text phrases are then annotated by the tag *explains*, which is corresponding to an object property of the upper ontology and can be exploited during problem–solving.

Integrating Knowledge The entered knowledge is committed to the knowledge wiki by simply saving the edit page using the mandatory "Save" button. Besides storing the textual content of the wiki page in the standard wiki repository, i.e., the text in natural language and the textual definitions of the knowledge, the knowledge wiki also extracts all knowledge–related parts of the article, i.e., the annotated text phrases and the knowledge bases in textual markup. The extracted parts are then compiled to an executable knowledge base corresponding to the considered wiki page, and are stored in the knowledge base repository along with other knowledge bases already compiled from other wiki pages. Figure 2 depicts this "Save and Integrate" workflow of the knowledge wiki. With the growth of the wiki the number of knowledge bases will also increase. The concepts used in the knowledge bases are naturally aligned with each other, if all knowledge bases reuse the pre–defined application ontology. However, for ad–hoc defined inputs and findings, respectively, we can easily express alignment rules, that match the concepts with concepts of the application ontology.

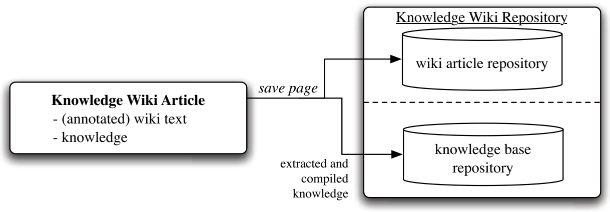


Figure 2: The "Save and Integrate" workflow of the knowledge wiki KNOWWE.

An Evolutionary Process Model In open and distributed environments, like the web–based knowledge engineering tool introduced here, an evolutionary process model has been found to be most suitable. For example, Fischer (Fischer 1998) proposes an evolutionary methodology that is based on the three phases *Seeding*, *Evolutionary Growth*, and *Reseeding*. Although, the process model was proposed for distributed design environments the concepts can be directly transfered to the collaborative development using knowledge wikis. As depicted in Figure 3 the project is started with a *seeding* phase, where domain specialists fill an empty system with an initial collection of knowledge bases. Thus, the knowledge wiki already has a utility for a selected subset of intended users. In the next phase (*evolutionary growth*), the system is used and extended by the domain specialists but also by users increasing the utility due to an increased availability of information and knowledge. At some point the reorganisation of the overall system becomes necessary when the utility starts to decrease due to unorganized content or redundant/inconsistent parts of the wiki's knowledge. Then, the *reseeding* phase is initialized, where domain specialists are performing refactorings and reorganisations within the entire knowledge wiki. The methodology is best applied if the tool provides efficient support for the evaluation and refactoring of the knowledge wiki. We discuss these issues in more detail in the concluding section.

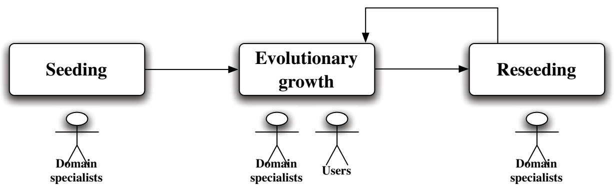
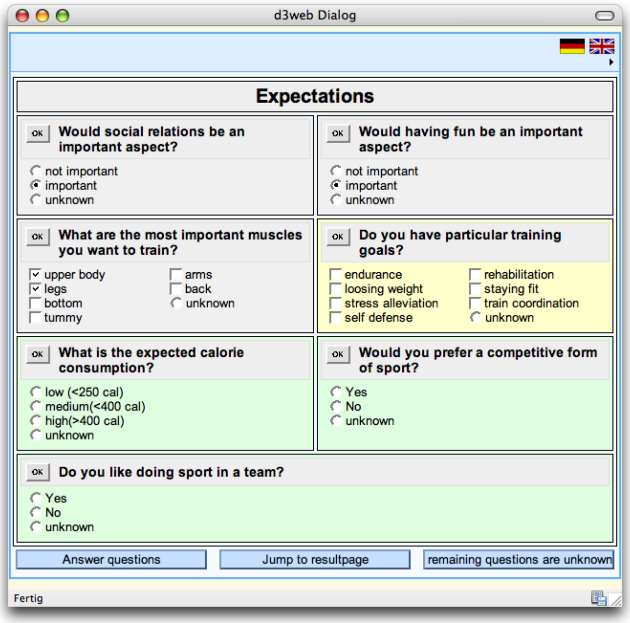


Figure 3: A collaboratiive process model: seeding, evolutionary growth, reseeding.

Wiki–based Knowledge Sharing

Concerning the current web and the task of finding a solution for a specific problem, we see that typically users are querying and browsing the web in order to retrieve an appropriate solution for a given problem. We call this procedure *manual problem–solving*, and we find many examples of knowledge clusters supporting this task. For example, bulletin boards and wikis are prominent sources of collaborative knowledge clusters, i.e., places where many people capture and share their knowledge. With semantic wikis capturing knowledge for particular domains this task is marginally simplified by improving the search by semantically meaningful concepts.

In knowledge wikis we aim to improve this manual problem–solving process by knowledge–based techniques: on the one hand, explicit problem–solving knowledge can be used in order to generate interactive interviews asking tailored questions that derive appropriate solutions. On the other hand, tagged text phrases are used to provide in–place answers, i.e., interactive elements in the standard wiki text that can be clicked to enter values for meaningful facts. Figure 1(b) shows the standard link for starting a generated interview, which opens a new dialog window with forms as shown in Figure 4. The use of in–place answers is shown at


d3web Dialog

Expectations

OK **Would social relations be an important aspect?**
not important
[✓ important]
unknown

OK **Would having fun be an important aspect?**
not important
[✓ important]
unknown

OK **What are the most important muscles you want to train?**
[✓ upper body]
[✓ legs]
bottom
tummy
arms
back
unknown

OK **Do you have particular training goals?**
endurance
loosing weight
stress alleviation
self defense
rehabilitation
staying fit
train coordination
unknown

OK **What is the expected calorie consumption?**
low (<250 cal)
medium(<400 cal)
high(>400 cal)
unknown

OK **Would you prefer a competitive form of sport?**
Yes
No
unknown

OK **Do you like doing sport in a team?**
Yes
No
unknown

Answer questions | Jump to resultpage | remaining questions are unknown

Fertig

Figure 4: An interactive interview generated by the knowledge wiki in order to collect findings.

Figure 1(c): a click on the text phrase ”endurance” opens a pop–up menue asking for the value of the concept *Training goals*, where ”endurance” is a possible value. Both ways to enter a finding in the knowledge wiki yield to the creation of a finding instance corresponding to the clicked finding. The instance is then propagated to the *knowledge wiki broker*, which is responsible for inferring solutions based on the given findings. The propagation paths of the broker are depicted in Figure 5. Here, every wiki page is wrapped in a *knowledge service*, that contains the corresponding knowledge base. We can see that the entered finding instances are propagated to the broker which itself aligns the findings to the globally known application ontology and then files the aligned instances to the central blackboard. When using a global application ontology this task is trivial, i.e., when every knowledge base of the wiki uses the same findings of the application ontology. Further, the broker notifies all connected knowledge bases of the existence of the new finding instance and enables the knowledge services to derive their solutions if possible. Potentially derived solutions are subsequently propagated to the broker again. By this means, entered findings are not only processed by the knowledge base of the current wiki page, but also by all existing knowledge bases of the wiki. Therefore, all solutions represented in the knowledge wiki can be derived at any page; already derived solutions are presented at the right pane of the wiki as for example shown in Figure 1(d). Here, the solutions ”Cycling” and ”Jogging” were derived as the most appropriate solutions, even though the findings were entered on the page concerning the solution ”Swimming”.

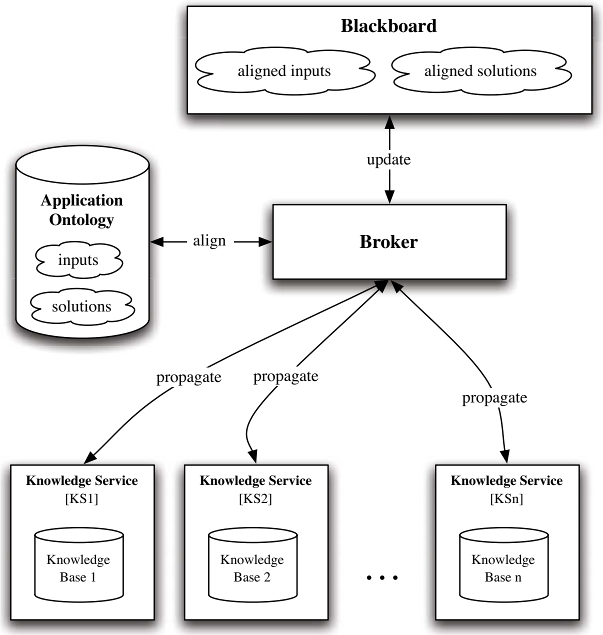


Figure 5: Blackboard architecture for the distributed problem–solving of the knowledge wiki KnowWE.

In this section, we briefly introduced the basic concepts of creating and sharing knowledge using a knowledge wiki. In the next sections, we describe the upper ontology for problem–solving in more detail, and we introduce knowledge markups for different knowledge representations and for annotating natural language text.

An Upper Ontology for Problem–Solving

The underlying knowledge representation of the knowledge wiki is given by a problem–solving ontology, that declaratively defines the basic concepts and their inter–relationships. This upper ontology denotes the fundament of every knowledge wiki application and defines the possible concepts and properties that can be instantiated for a specific knowledge wiki application. Like in semantic wikis, additional ad–hoc concepts and properties can be defined during the use of the knowledge wiki; however, ad–hoc additions not necessarily have implications for the problem–solving process.

Concepts and Properties In the following we briefly describe the relevant concepts and properties of the upper ontology, and we explain their meaning for the problem–solving process. In Figure 6 the most relevant parts of the upper ontology are shown and we identify the concept *Finding* [1] as the central object of the knowledge representation. A finding has three important relations defined by the properties *hasInput*, *hasValue*, and *hasComparator*: The first two properties are used to express, that a finding holds a value assigned to a (user) input, e.g., in the context of the sports advisor example the finding *Training goals = endurance* the value *endurance* was assigned to the input *Training goals*. The actual assignment (and also the comparison for later tasks) is described by the property *hasComparator*; in our example the equals comparator $=$ was applied. We distinguish findings by their type of values that can be assigned to the input, i.e., numeric, predefined choice values or text strings. The relations of the subclasses of *Finding* are constrained by property restrictions to proper subclasses of *Input* [2] , *Value* [3], and *Comparator* [4]. The property restrictions are not visualized in the figure.

It is interesting to see that solutions of the system are represented by the concept *Solution* [5] as a subclass of one–choice inputs, i.e., inputs that have a predefined value range. For solutions the value range is restricted to the exhaustive set of subclasses of *Value_Solution* representing the states $\{$*derived, suggested, excluded*$\}$. In order to enable a suitable interview structure, the user is able to group meaningful inputs by defining subclasses of the concept *Questionnaire*. For the sports advisor example, we would define distinct questionnaires capturing user requirements (e.g., training goals) and user restrictions (e.g., medical restrictions). The relation between a questionnaire and its contained inputs is described by the property *groupsInputs*.

A concrete problem–solving session is then represented by the concept *PSSession* [7] holding a number of findings (represented by the property *storesFindings*) that were entered by the user as well as derived by the system. In the following sections we describe the semantic annotation of findings by the object property *explains*: the property is defined between a finding (mostly a solution) as the domain and other findings as the range. Using this property we are able to define findings that are responsible for deriving a particular *solution*.

Building a new Application Every knowledge wiki defines new concepts for inputs, findings, and solutions, respectively, describing the implemented application domain. The concrete inputs and solutions are then representing the *application ontology* by subclassing the upper ontology sketched above. For example, in the sports advisor wiki the user input *Training goals* is a subclass of *Input_Multiple_Choice* having the possible values *endurance*, *lossing weight* etc. (with multiple–choice inputs, we allow users to specify more than one training goal). For a structured knowledge acquisition process it is usually reasonable to propose a terminology of findings that is (re)used for all pages of the knowledge wiki, i.e., the derivation knowledge for the solutions makes use of a pre–defined set of findings. A terminology can be created and managed by providing a distinct wiki page, i.e., *WikiFindings*, that captures all relevant questionnaires, their included inputs and values, respectively. The following example shows excerpts of the textual markup of the sports advisor application ontology. A simple textual markup is used that allows users to rapidly create new concept definitions and to structure these in a taxonomy. Subclasses between concepts of a previous line and the concept of the current line are denoted by a hyphen ("-") sign. For the definition of questionnaires, we interpret concepts with no leading hyphen as direct children of the upper ontology concept *Questionnaire*. Findings are directly related to a concrete questionnaire concept. The markup `<Questionnaire-section>` is used to define a taxonomy of questionnaires; here, we defined (implicitly) *Advisor questions* to be a subclass of *Questionnaire* (line 2), whereas *Expectations* and *Restrictions* are subclasses of *Advisor questions* (lines 3–4).

```
1  <Questionnaire-section>
2  Advisor questions
3  - Expectations
4  - Restrictions
5  </Questionnaire-section>
6
7  <Trees-section>
8  Expectations
9  - Social interaction [oc]
10 -- wanted
11 -- unimportant
12 - Muscles [mc]
13 -- upper body
14 -- legs
15 -- arms
16 -- tummy
17 -- back
18 - Training goals [mc]
19 -- endurance
20 -- loosing weight
21 -- stress alleviation
22 ...
23 </Trees-section>
```

Furthermore, inputs and their corresponding values are directly added to defined questionnaires in the paragraph `<Trees-section>` . For example, in lines 9–11 the one–choice input concept *Social interaction* is defined with the two possible value concepts *wanted* and *unimportant*; thus defining the two findings *Social interaction = wanted* and *Social interaction = unimportant*. For the given input concepts a *groupsInputs* relation between the named questionnaire and the inputs is created.

As we have seen, we need to define (at least a seed of) findings before filling the wiki with solutions and derivation knowledge. In contrast, new solutions are usually added it-

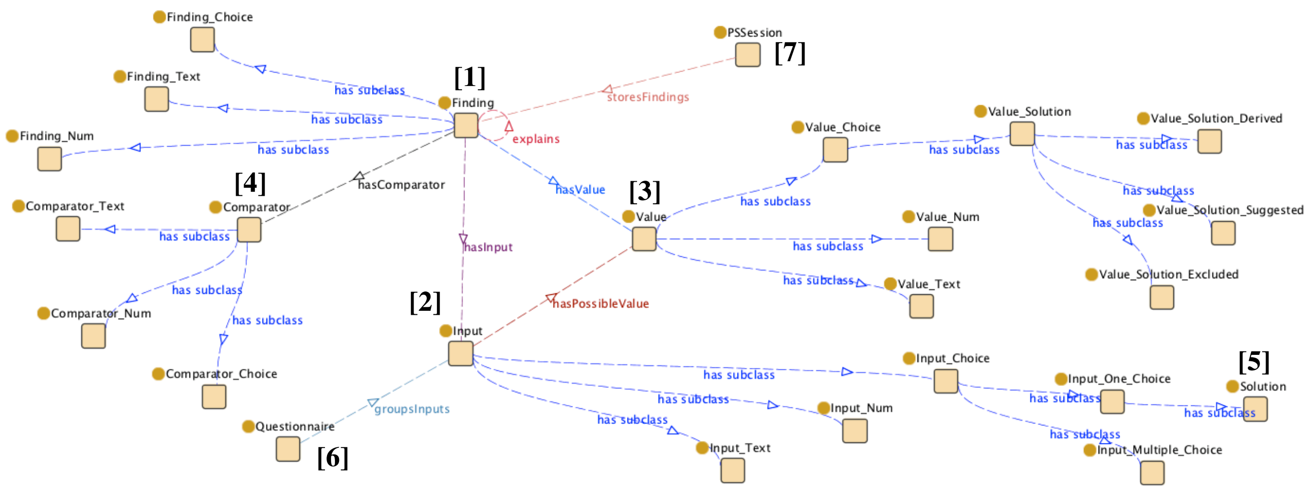


Figure 6: An excerpt of the upper ontology describing the relevant concepts and properties in a knowledge wiki.

eratively to the system by simply creating a new wiki page for the solution. By default, the new solution is added as a subclass of the concept *Solution*, if not defined as a subclass of another solution in the special wiki page *WikiSolutions*. This special page provides the interface to define a solution hierarchy of the application ontology in a textual manner, which is tagged by `<Solutions-section>`.

```
1  <Solutions-section>
2  Sport
3  - Water sports
4  -- Swimming
5  ...
6  - Endurance sports
7  -- Running
8  -- Swimming
9  ...
10 </Solutions-section>
```

Similar to the definition of questionnaires, concepts with no preceding hyphen are interpreted as direct subclasses of the upper ontology concept *Solution*, e.g. the concept *Sport* in line 2. Further subclasses are created by an increased indent of hyphens, e.g., *Water sports* (line 3) and *Endurance sports* (line 6) as a subclasses of *Sport*, and *Swimming* in line 4 as a subclass of *Water sports*. It is interesting to see that a poly–hierarchy can be defined by multiple inheritance, e.g., *Swimming* is defined as a subclass of *Water sports* and *Endurance sports*.

The dynamic behavior of the wiki and the knowledge bases is represented by an instance of the concept *PSSession*. By holding multiple instances of the property *hasFinding* we are able to capture all entered finding instances that are provided by a particular user. If a user enters a finding, then a new instance of this finding is created and linked to the corresponding *PSSession* instance by the *hasFinding* property. These instances are then used during the distributed problem–solving process as shown in Figure 5.

Markups for the Definition of Problem–Solving Knowledge

In the previous section we introduced the upper ontology of the knowledge wiki and we showed how this ontology can be extended by an application ontology layer. This section introduces a variety of markups to define problem–solving knowledge, i.e., connecting defined findings with appropriate solutions. To come up with the different requirements of the particular application projects we provide alternative possibilities to capture the derivation knowledge in the wiki. In any case, the syntax of the knowledge markups should be as simple as possible in order to allow for an intuitive creation and evolution of the knowledge together with the normal wiki text. In the best case, typical wiki users are capable to understand and use the knowledge wiki syntax without a thorough training.

Set–Covering Knowledge

Set–covering models (Reggia, Nau, & Wang 1983) are an intuitive representation to express abductive knowledge for given solutions. For a given solution a model then describes a list of findings, that are usually observed when the solution is present. During the problem–solving process the user enters findings into the system and the reasoner is selecting a list of best matching solutions, i.e., solutions with a maximal intersection of expected and entered findings. Besides such a simple list of expected findings a set–covering model can be incrementally extended by background knowledge to improve its expressiveness (Baumeister, Seipel, & Puppe 2003).

To be used in the context of the knowledge wiki we dramatically simplified the knowledge representation to a set of findings for each solution. For example, the markup below shows a part of the set–covering listing for the solution *Running*, where the name of the solution is followed by textual descriptions of findings listed in braces ($\{\ldots\}$). The set is not meant to be a conjunction or disjunction of expected findings, but as a flat collection of possibly occurring findings for the given solution.

```
<SetCoveringList-section>
  Running {
    Muscles = legs,
    Training goals = loosing weight,
    Training goals = endurance,
    Social interaction = unimportant,
    ...
  }
</SetCoveringList-section>
```

In the simplest version, the problem solver intersects the set of observed findings with the set of expected findings for each solution and returns the best matching solutions.

Users can rapidly create (flat and simple) knowledge for a couple of solutions in order to make experience with the features of the knowledge wiki. For some of the solutions the set–covering knowledge may remain during further development; for other solutions the users will aks for a more expressive knowledge representation. For these solutions we offer alternative representations such as rules or decision trees, that we describe in the following paragraphs.

In the past, we have made very positive experiences with the use of set–covering lists as a "starter knowledge representation" and a conditional switch to more complex representations. Commonly, users were motivated at the beginning of a project, since the set–covering knowledge representation was so simple, that even untrained users were able to immediately understand and to apply it to their application domain. Later in the project, when they saw the requirements for a stronger expressiveness, they were willing to get training on the semantics and use of rules or decision trees.

Rules

Rules have been the most popular knowledge representation over the last decades, and rule–based systems are known to be used in many application domains.

A rule $r = c_r \rightarrow a_r$ derives facts as defined in its consequent (rule action) a_r, if the specified rule condition c_r is satisfied. Entered and already derived facts can serve as final solutions presented to the user, but can be also used as further input to conditions of other rules. In a general setting a rule condition is defined by nested (negated) conjunctions and disjunctions of constrained user inputs.

Definition and Derivation All explicit rules of a wiki page are defined in a textual manner within the `<Rules-section>` tag. For example, the following markup specifies a rule base with one rule, that computes the value of the body-mass index BMI. The result of the computation is assigned to the numerical input *BMI*, if the numerical inputs *Height* and *Weight* have both values greater than 0. It is worth noticing, that abstraction rules as shown above can be easily applied to express knowledge for matching between different findings (*ontology matching*).

```
<Rules-section>
 // A comment for abstraction rule
 IF   (Height > 0) AND (Weight > 0)
 THEN BMI = (Weight / (Height * Height))
</Rules-section>
```

Besides abstraction rules we also provide a rule representation for the graded derivation of solutions using scoring weights. Scores are used to represent a qualitative approach for deriving solutions having a scoring weight. These weights can have a positive or negative sign, and state the degree of confirmation or disconfirmation of a particular solution. The definition and semantics of scoring rules goes back to the INTERNIST/QMR project (Miller, Pople, & Myers 1982). The use of score weights is simplified by introducing symbolic categories for positive and negative scoring weights. 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, and 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. The following markup defines a rule base with two scoring rules; rule $r1$ adds the symbolic weight P5 to the score of the solution instance of *Running*, if all three sub–conditions hold. Scoring rule $r2$ additionally adds the symbolic weight P2 to the score, if the input *Costs* was answered with value *low*.

```
<Rules-section>
 // A comment for scoring rule r1
 IF  ("Training goals" = endurance)
     AND NOT("BMI" > 30)
     AND ("Physical Problems" = knees)
 THEN Running = P5

 // A comment for scoring rule r2
 IF   Costs = low
 THEN Running = P2
</Rules-section>
```

During a problem–solving session score weights of fired rules are aggregated, and a solution is derived and presented to the user, when the aggregated score exceeds a given threshold; for the presented knowledge representation the threshold is set to P5. The order of the rules listed in the rule base has no defined meaning, e.g., with respect to a conflict resolution strategy of rule engines.

Alternative Markups In the context of knowledge wikis the simplicity and clearness of the markup is essential for the success and its application, respectively. Therefore, we decided to *not* use already existing standard markups for (horn clause) rules like SWRL/RuleML (Horrocks *et al.* 2005), but to promote a more compact and human–readable notion as sketched above.

Decision Trees

Decision trees are also a popular and intuitive representation for building knowledge systems manually. They represent knowledge for the derivation of solutions but also the order of relevant inputs to be asked to the user in a problem–solving session. The markup of decision tree knowledge is similar to the definition of the application ontology of findings, where the user inputs are defined in coherent groups of questions. For decision trees we interpret these groups as questionnaires, that follow a given interview structure. Here, every questionnaire represents a single decision tree, and every leaf of a tree can either indicate the activation of another decision tree or derive a solution.

Definition and Derivation All decision trees of a wiki page are defined within the `<Trees-section>` tag. The interview structure is specified by the use of hyphens (“`-`”) as described in the following. The root of a particular tree is not preceded by a hyphen and specifies the name of the decision tree. The following line of the tree is indented by one hyphen and represents the first user input to be asked, i.e., the first question. The following lines are indented by an increased count of hyphens and usually state the possible answers of the preceding question. In decision trees, follow–up questions are presented depending on the previously given answers. In the textual markup we represent follow–up questions by defining the questions and there subsequent answers by defining them with an increased indent in the line next after the depending answer. The following example shows the two decision trees *Naive Sports Advisor* and *Muscle Questions* (the line numbers are only added for describing the particular lines).

```
1  <Trees-section>
2  Naive Sports Advisor
3  - Training goals [oc]
4  -- endurance
5  --- Physical problems [oc]
6  ---- knees
7  ----- Swimming (!)
8  ---- no problems
9  ----- Running (!)
10 -- increase muscles
11 --- Muscle Questions
12
13 Muscle Questions
14 - Favorite regions of muscles [mc]
15 -- arms
16 ...
17 </Trees-section>
```

We see that the two decision trees are defined in line 2 and 13 by having no preceding hyphen. The first question of *Naive Sports Advisor* is the one–choice question *Training goals* already defined in the ontology of user inputs. The type of the question is additionally specified by `[oc]` at the end of the line. Besides one-choice questions we are also free to define multiple–choice questions `[mc]` and numeric questions `[num]`. The follow–up input *Physical problems* is presented to the user, it the value *endurance* was given. Further on, the solution *Swimming* is derived in line 7, if the value *knees* is given. The markup `(!)` succeeding the entry *Swimming* specifies the categoric derivation of the solution, i.e., an instance of *Value_Solution_Derived* is assigned to the solution instance of *Swimming*. Also, the solution *Running* is derived in line 9, if the input *Physical problems* was answered with the value *no problems*.

Modularization, Interconnection and Reuse Large decision trees are inflexible and hardly maintainable especially using such a textual representation. Therefore, the modularization and simplification of the knowledge into small chunks of decision trees is a common and reasonable procedure, as for example proposed by the knowledge formalization pattern *heuristic decision tree* (Puppe 2000). A decision tree is then connected by the activation of other decision trees as a possible action in its leafs. In line 11 the decision tree *Muscle Questions* is activated for presentation, if the previous input *Training goals* was answered with the value *increase muscles*.

Usually, decision trees require an interactive interview with the user which can be activated by the default ”Interview” link placed on every wiki page. However, the derivation knowledge defined in decision trees can be also used by the facts entered by in–place answers, since we encode the derivation knowledge of decision trees by ordinary rules. For example, the derivation knowledge of the decision tree *Naive Sports Advisor* is encoded by the following rules.

```
<Rules-section>
 IF  (Training goals = endurance)
      AND (Physical Problems = knees)
 THEN Swimming = P7

 IF  (Training goals = endurance)
      AND (Physical Problems = no problems)
 THEN Running = P7
</Rules-section>
```

Alternative Markups It is easy to see that such a decision tree structure could be simply formulated using a special XML markup as suggested in the following.

```
<tree id="NaiveSportsAdvisor">
  <input name="Training goals"
         type="oc">
    <value name="endurance">
       <input name="Physical problems"
              type="oc">
         <value name="knees">
           <solution name="Swimming"
                     weight="!" />
         </value>
         <value name="no problems>
           <solution name="Running"
                     weight="!" />
         </value>
       </input>
    </value>
    <value name="increase muscles>
      <ref refid="MuscleQuestions"/>
    </value>
  </input>
</tree>
```

The use of an XML markup would allow for the application of XML tools and would reduce the compilation effort. More importantly, the reuse of the knowledge would be increased due to the use of a standardized textual syntax. However, the application of such a verbose syntax would be too complex and error-prone to be used by knowledge wiki engineers.

Markup for Inline Knowledge Annotation

As described before, a wiki page typically describes the information on and the knowledge deriving a particular solution. Besides normal text in natural language the knowledge should be also represented in a more explicit, machine–interpretable manner. Like in semantic wikis we provide methods to semantically annotate text phrases with ontological concepts. In previous work, knowledge wikis were only able to tag specific text phrases with a reference to user inputs (Baumeister, Reutelshoefer, & Puppe 2007), and the annotation was merely applied for enabling the in–place answers technique. This paper describes an extended approach, where text phrases in natural language are not only tagged by user inputs but also entail the definition of derivation knowledge using the ontologies defined above.

Annotation Syntax The proposed knowledge annotation uses square brackets to relate text phrases with ontological concepts. In general, flat derivation knowledge between the corresponding solution of the wiki page and a particular finding is defined by the associative relation *explains*; this relation was defined as an object property with a finding (mostly a derived value assigned to a solution) as the domain and a standard finding as the range.

The following definition specifies the general markup of the proposed knowledge annotation (underlined characters are optional):

$$\textbf{[}\underline{\textbf{text <=>}}\ \textbf{explains::}\underline{\{\textbf{finding,}\}^*}\ \textbf{finding]}$$

If $\underline{\textbf{text <=>}}$ is used, then the phrase `text` is displayed in the view mode of the wiki page and a link for in–place answers is available on `phrase`. Otherwise, the word immediately preceding the particular annotation is assumed to be used for the in–place answers link. The mandatory token `explains::` creates new property instances of the object property *explains* between an solution instance representing the wiki page and the findings specified after `explains::`; here, at least one finding needs to be defined but more than one findings are allowed for creating multiple property instances at once. The specified property instances can be used as flat derivation knowledge.

For example, the following phrase in the edit pane of the wiki page *Swimming* defines associative relations between the solution *Swimming* and values of the user input *Training goals*.

```
Swimming is good for training the endurance
[explains::Training Goals = endurance]
or for successfully [reducing weight <=>
explains::Training Goals = reducing weight].
```

The first annotation results in an in–place answer link on *endurance* and creates a new relation for *Swimming* and *Training Goals = endurance*; here, the directly preceding word is used for the link. The second annotation displays *reducing weight* as in–place answers link and creates a relation for *Swimming* and *Training Goals = reducing weight*. In the presented implementation we interpret the *explains* relation as a set–covering relation between the *derived* value of the specified solution and the specified finding instance. Therefore, the example annotations from above are equivalent to the following set–covering list:

```
Swimming {
  Training Goals = endurance,
  Training Goals = reducing weight
}
```

Thus, we are able to use inline knowledge annotations as an alternative of the explicit specification of set–covering knowledge.

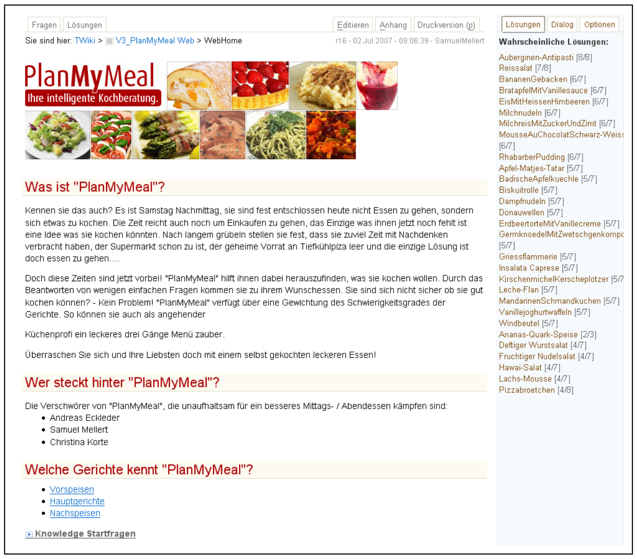
Discussion Using knowledge annotations, explicit problem–solving knowledge can be seamlessly combined with the standard wiki text given natural language. The proposed syntax follows the markup of Semantic MediaWiki (Krötzsch, Vrandečić, & Völkel 2006) for the definition of property instances. We mainly introduce by the optional marker `text <=>` to allow for in–place answers. Due to the existence of the upper ontology the property *explains* has a specific, predefined semantics that is interpreted as a set–covering relation.

Case Studies

Although, the presented knowledge wiki KnowWE is lively under development and the parts concerning the functionality of a semantic wiki are currently improved, we report on experiences we have made up to now in different projects.

General Recommender Wikis

This case study was intended as a proof–of–concept project considering the development of a larger knowledge wiki involving people initially unfamiliar with the system. Here, 45 students formed 11 groups, where each group was responsible for the development and the evolution of one knowledge wiki. The groups were free to choose an application domain they were already familiar with. For example, knowledge wikis for meal selection, recommenders for holidays and recreation, a movie advisor, and a wine selection were started. At the project start the previously untrained students were introduced into the concepts of the knowledge wiki and its markup (took about 1h). After that the students built the initial versions of their systems. Figure 7 shows a screen shot of the (german) system "PlanMyMeal": the recommendation system proposes appropriate meals (solutions) based on the user preferences and the ingredients currently available in the household (findings).



Sie sind hier: TWiki > V3_PlanMyMeal Web > WebHome

r16 - 02 Jul 2007 - 09:06:39 - SamuelMellert

Was ist "PlanMyMeal"?

Kennen sie das auch? Es ist Samstag Nachmittag, sie sind fest entschlossen heute nicht Essen zu gehen, sondern sich etwas zu kochen. Die Zeit reicht auch noch um Einkaufen zu gehen, das Einzige was ihnen jetzt noch fehlt ist eine Idee was sie kochen könnten. Nach langem grübeln stellen sie fest, dass sie zuviel Zeit mit Nachdenken verbracht haben, der Supermarkt schon zu ist, der geheime Vorrat an Tiefkühlpiza leer und die einzige Lösung ist doch essen zu gehen....

Doch diese Zeiten sind jetzt vorbei! "PlanMyMeal" hilft ihnen dabei herauszufinden, was sie kochen wollen. Durch das Beantworten von wenigen einfachen Fragen kommen sie zu ihrem Wunschessen. Sie sind sich nicht sicher ob sie gut kochen können? - Kein Problem! "PlanMyMeal" verfügt über eine Gewichtung des Schwierigkeitsgrades der Gerichte. So können sie auch als angehender

Küchenprofi ein leckeres drei Gänge Menü zauber.

Überraschen Sie sich und Ihre Liebsten doch mit einem selbst gekochten leckeren Essen!

Wer steckt hinter "PlanMyMeal"?

Die Verschwörer von "PlanMyMeal", die unaufhaltsam für ein besseres Mittags- / Abendessen kämpfen sind:

- Andreas Eckleder
- Samuel Mellert
- Christina Korte

Welche Gerichte kennt "PlanMyMeal"?

- Vorspeisen
- Hauptgerichte
- Nachspeisen

Knowledge Startfragen

Wahrscheinliche Lösungen:

Auberginen-Antipasti [8/8]
Reissalat [7/8]
BananenGebacken [6/7]
BratapfelMitVanillesauce [6/7]
EisMitHeissenHimbeeren [6/7]
Milchnudeln [6/7]
MilchreisMitZuckerUndZimt [6/7]
MousseAuChocolatSchwarz-Weiss
[6/7]
RhabarberPudding [6/7]
Apfel-Matjes-Tatar [5/7]
BadischeApfelkuechle [5/7]
Biskuitrolle [5/7]
Dampfnudeln [5/7]
Donauwellen [5/7]
ErdbeertorteMitVanillecreme [5/7]
GermknoedelMitZwetschgenkompo
[5/7]
Griessflammerie [5/7]
Insalata Caprese [5/7]
KirschenmichelKerscheplotzer [5/7
Leche-Flan [5/7]
MandarinenSchmandkuchen [5/7]
Vanillejoghurtwaffeln [5/7]
Windbeutel [5/7]
Ananas-Quark-Speise [2/3]
Deftiger Wurstsalat [4/7]
Fruchtiger Nudelsalat [4/7]
Hawai-Salat [4/7]
Lachs-Mousse [4/7]
Pizzabroetchen [4/8]

Figure 7: A recommender system for proposing suitable meals based on user preferences and ingredients currently available at home.

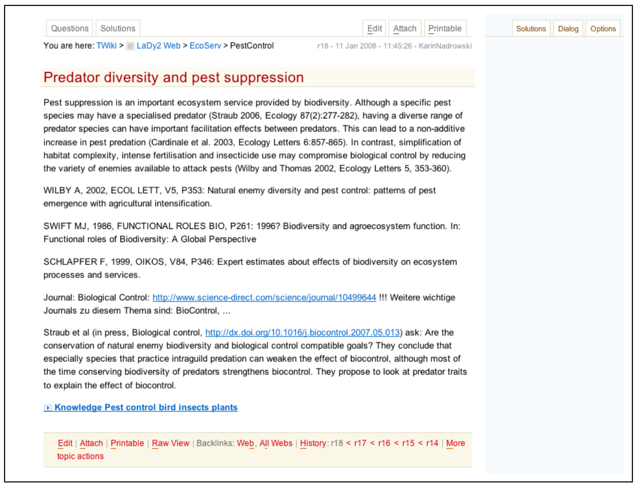
In general, the set–covering list appeared to be the predominant knowledge representation, since it was experienced to be intuitive and sufficient in most of the cases. A significant disadvantage of set–covering models are their inability to represent negative (exclusion) knowledge, i.e., the presence of user inputs for which a solution should be discarded from the recommendations. This problem was tackled by the usage of a hybrid rule–extension: here, the users are able to define rule conditions under which a solution is excluded even when the result of the set–covering inference was positive.

In total, the case study was finished successfully with about 700 knowledge bases distributed over 11 knowledge wikis. During the last two thirds of the project we were able to count about 2500 edits of the particular knowledge bases and their wiki pages, respectively. Due to a failure the logs were not available for the first third of the case study.

LaDy – Landscape Biodiversity

The second case study reported here considers a real–world problem integrating domain specialists from the biological domain. In the last years, landscape ecologists have collected a large amount of (unstructured) knowledge on *landscape diversity of life* with respect to the given landscape structures, management decisions and their progression (Otte, Simmering, & Wolters 2007). Inter– and Trans–disciplinary research projects with economists yielded socioeconomic knowledge on how the biodiversity can be supported in managed agro–ecosystems. However, the collected knowledge is commonly not directly accessible to decision makers in private and governmental agencies. For this reason, one purpose of the BIOLOG Europe project (http://www.biolog-europe.org) is the integration of socioeconomic and landscape ecological research results in order to produce an common understanding of the effects of environmental change on managed ecosystems.

In this context, the knowledge wiki LaDy (for ”Landscape Diversity”) supports domain specialists as well as related people to collect and share knowledge at different levels ranging from textual descriptions and multimedia to explicit knowledge bases covering the effects on landscape diversity. Solutions are defined concerning the biodiversity of various taxa, different ecosystem services and management decisions. At the moment, the knowledge wiki is under development incorporating ecological domain specialists distributed all over germany. For example, Figure 8 shows a wiki page collecting relevant information on predator diversity and their effects on pest suppression. Besides summarizing the effects and citing the relevant literature on this topic, some knowledge on the effects of pest suppression with respect to predator diversity is modelled. Providing a



You are here: TWiki > LaDy2 Web > EcoServ > PestControl

r18 - 11 Jan 2008 - 11:45:26 - KarinNadrowski

Predator diversity and pest suppression

Pest suppression is an important ecosystem service provided by biodiversity. Although a specific pest species may have a specialised predator (Straub 2006, Ecology 87(2):277-282), having a diverse range of predator species can have important facilitation effects between predators. This can lead to a non-additive increase in pest predation (Cardinale et al. 2003, Ecology Letters 6:857-865). In contrast, simplification of habitat complexity, intense fertilisation and insecticide use may compromise biological control by reducing the variety of enemies available to attack pests (Wilby and Thomas 2002, Ecology Letters 5, 353-360).

WILBY A, 2002, ECOL LETT, V5, P353: Natural enemy diversity and pest control: patterns of pest emergence with agricultural intensification.

SWIFT MJ, 1986, FUNCTIONAL ROLES BIO, P261: 1996? Biodiversity and agroecosystem function. In: Functional roles of Biodiversity: A Global Perspective

SCHLAPFER F, 1999, OIKOS, V84, P346: Expert estimates about effects of biodiversity on ecosystem processes and services.

Journal: Biological Control: http://www.science-direct.com/science/journal/10499644 !!! Weitere wichtige Journals zu diesem Thema sind: BioControl, ...

Straub et al (in press, Biological control, http://dx.doi.org/10.1016/j.biocontrol.2007.05.013) ask: Are the conservation of natural enemy biodiversity and biological control compatible goals? They conclude that especially species that practice intraguild predation can weaken the effect of biocontrol, although most of the time conserving biodiversity of predators strengthens biocontrol. They propose to look at predator traits to explain the effect of biocontrol.

Knowledge Pest control bird insects plants

Edit | Attach | Printable | Raw View | Backlinks: Web, All Webs | History: r18 < r17 < r16 < r15 < r14 | More topic actions

Figure 8: A wiki page of the LaDy wiki concerning predator diversity and pest suppression.

platform for both, exchanging textual knowledge and implementing explicit rules on ecosystem behaviour, LaDy provides a service to condense and to communicate knowledge needed for an efficient management of ecosystem services provided by biodiversity.

Up to now, almost every user of the wiki started with set–covering lists as the initial knowledge representation. As it worked out for many simple relations between the modeled solutions and the findings, knowledge bases of more complex solutions are currently refactored to a rule–based representation.

Conclusions

We have introduced a novel platform for web–based knowledge engineering using semantic wikis extended by different types of problem–solving knowledge.

Summary

Like in many semantic wiki implementations the normal text is semantically annotated relating natural language text with ontological concepts. In addition, we provide specific properties like *explains*, that are defined by the *upper ontology* for problem–solving, and that have a distinct semantics for the problem–solving task. Whenever a user annotates text using the knowledge wiki markup, this new knowledge is added to a knowledge base. In general, every wiki page holds a separate knowledge base containing all the annotated relations and knowledge markups. A distributed problem–solving process is applied to facilitate knowledge sharing across the contents of the knowledge wiki.

The sketched approach offers some advantages, that are often not handled satisfactory in traditional knowledge engineering approaches: Using a standard wiki interface many users are able to immediately work together in a problem domain, since wikis denote an established technology that is already known. Furthermore, it becomes very easy to mix different granularities of knowledge, e.g., by using standard wiki text as the simplest and least explicit form of knowledge and textually markup-ed rules as the most explicit representation.

Research Directions

However, the application of a new technology usually brings some challenges that cannot be handled with existing methods. Most of the challenges are grounded in the distributed knowledge formalization at different levels of detail using a web–based system. In the following, we sketch the most relevant research directions that are motivated by the experiences we made in the projects.

Distributed Refactoring As described before, we propose to start a project with the simplest knowledge representation possible and then switch to more expressive representations when needed. Although, the system supports reasoning with knowledge bases using varying representations, the actual "switch" of a representation concerning an existing knowledge base denotes a complex transformation task. In Software Engineering such a transformation is called *refactoring*, i.e., changing the design of a piece of software without modifying the intended semantics (Fowler 1999). Refactoring methods in the context of knowledge engineering research has been investigated in the past, e.g., (Gil & Tallis 1997; Baumeister, Seipel, & Puppe 2004), but only considered the refactoring of a separate knowledge base without considering implications for other knowledge bases. Small refactorings like *Rename Concept* are already implemented in the system, and are successfully applied to unify concepts with different names but identical meaning. Larger refactorings like the transformation of one knowledge representation such as set–covering lists to a rule base are an open issue for future work. Preliminary investigation has been done by analyzing refactorings to knowledge formalization patterns (Halstead 2005). A technical problem will arise when the interface of complex refactoring methods are implemented in the context of a web–based environment such as wikis, since this kind of refactoring usually requires frequent user interaction.

Evaluation We distinguish three subtasks that are relevant for the evaluation of knowledge wiks: the design analysis, verification, and validation of knowledge wikis. In the context of collaborative knowledge engineering projects the actual *design* of the evolving knowledge base is of prime interest. For knowledge wikis we investigate the design of either a single knowledge base of a distinct wiki page or we can analyse the overall design of the (distributed) knowledge base generated by all knowledge bases contained in the wiki. For individual knowledge bases some design anomalies were already proposed, e.g., (Baumeister & Seipel 2005). When considering multiple knowledge bases we also need to take the heterogenous level of detail of the formalized knowledge into account.

The *verification* of knowledge bases also considers formal methods for anomaly detection for individual knowledge bases as well as for distributed knowledge bases. For individual knowledge bases a large amount of research is already available, e.g., (Preece & Shinghal 1994) for classic rule systems and, e.g., (Baumeister & Seipel 2006) for verification of rule bases mixed with ontologies. However, less attention was paid on the verification of distributed knowledge bases. Furthermore, the application of knowledge wikis additionally adds another interesting issue for verification: often, the formalized knowledge is also redundantly represented in natural language text next to the knowledge base specification. Matching natural language text with the formalized knowledge and identifying contradicting or missing information may be an challenging task for future research, and will certainly build on ontology learning methods (Buitelaar, Cimiano, & Magnini 2005).

Similarly, the *validation* of knowledge wiki systems should consider the validation of single knowledge bases as well as the validation of the entire system, i.e., the distributed knowledge base. Here, for example empirical testing methods need to be refined in the context of the distributed setting. Another point considers the content (text and knowledge) validation by collaborative user validation. Then, the utility and appropriateness of the content can be evaluated by user assessments, e.g., user rating of websites, as already done by many open web systems.

All the issues described above will open interesting directions for future research.

References

Auer, S.; Dietzold, S.; and Riechert, T. 2006. OntoWiki – A Tool for Social, Semantic Collaboration. In *ISWC'06: Proceedings of the 5th International Semantic Web Conference*, 736–749. Berlin: Springer.

Baumeister, J., and Seipel, D. 2005. Smelly Owls – Design Anomalies in Ontologies. In *FLAIRS'05: Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference*, 215–220. AAAI Press.

Baumeister, J., and Seipel, D. 2006. Verification and Refactoring of Ontologies With Rules. In *EKAW'06: Proceedings of the 15th International Conference on Knowl-*

edge Engineering and Knowledge Management, 82–95. Berlin, Germany: Springer.

Baumeister, J.; Reutelshoefer, J.; and Puppe, F. 2007. Markups for Knowledge Wikis. In *SAAKM'07: Proceedings of the Semantic Authoring, Annotation and Knowledge Markup Workshop*, 7–14.

Baumeister, J.; Seipel, D.; and Puppe, F. 2003. 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.

Baumeister, J.; Seipel, D.; and Puppe, F. 2004. Refactoring Methods for Knowledge Bases. In *EKAW'04: Engineering Knowledge in the Age of the Semantic Web: 14th International Conference, LNAI 3257*, 157–171. Berlin, Germany: Springer.

Buitelaar, P.; Cimiano, P.; and Magnini, B. 2005. *Ontology Learning from Text: Methods, Evaluation and Applications*, volume 123 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Fischer, G. 1998. Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain–Oriented Design Environments. *Automated Software Engineering* 5(4):447–464.

Fowler, M. 1999. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley.

Gil, Y., and Tallis, M. 1997. 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*, 377–383. AAAI Press.

Halstead, S. 2005. Refactoring to Knowledge Formalization Patterns. Master's thesis, University of Würzburg, Germany.

Horrocks, I.; Patel-Schneider, P. F.; Bechhofer, S.; and Tsarkov, D. 2005. OWL Rules: A Proposal and Prototype Implementation. *Journal of Web Semantics* 3(1):23–40.

Krötzsch, M.; Vrandecić, D.; Völkel, M.; Haller, H.; and Studer, R. 2007. Semantic Wikipedia. *Journal of Web Semantics* 5(4):251–261.

Krötzsch, M.; Vrandecić, D.; and Völkel, M. 2006. Semantic MediaWiki. In *ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273*, 935–942. Berlin: Springer.

Miller, R. A.; Pople, H. E.; and Myers, J. 1982. INTERNIST-1, an Experimental Computer-Based Diagnostic Consultant for General Internal Medicine. *New England Journal of Medicine* 307:468–476.

Otte, A.; Simmering, D.; and Wolters, V. 2007. Biodiversity at the Landscape Level: Recent Concepts and Perspectives for Multifunctional Use. *Landscape Ecology* 22:639–642.

Preece, A., and Shinghal, R. 1994. Foundation and Application of Knowledge Base Verification. *International Journal of Intelligent Systems* 9:683–702.

Puppe, F. 2000. Knowledge Formalization Patterns. In *Proceedings of PKAW 2000*.

Reggia, J. A.; Nau, D. S.; and Wang, P. Y. 1983. Diagnostic Expert Systems Based on a Set Covering Model. *Journal of Man-Machine Studies* 19(5):437–460.

Schaffert, S. 2006. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *STICA'06: 1st International Workshop on Semantic Technologies in Collaborative Applications*.