Towards Meta-Engineering for Semantic Wikis

Jochen Reutelshoefer, Joachim Baumeister, Frank Puppe

Institute of Computer Science, University of Würzburg, Germany
{lastname}@informatik.uni-wuerzburg.de

Abstract. Building intelligent systems is a complex task. In many knowledge engineering projects the knowledge acquisition activities can significantly benefit from a tool, that is tailored to the specific project setting with respect to domain, contributors, and goals. Specifying and building a new tool from scratch is ambitious, tedious, and delaying. In this paper we introduce a wiki-based meta-engineering approach allowing for the smooth beginning of the knowledge acquisition activities going along with tool specification and tailored implementation. Meta-engineering proposes that in a wiki-based knowledge engineering project not only the content (the knowledge base) should be developed in evolutionary manner but also the tool itself.

1 Introduction

The development of knowledge-based systems still suffers from the *Knowledge Acquisition Bottleneck* yielding high development costs with respect to the knowledge acquisition efforts. Usually, a knowledge acquisition tool poses several constraints to the engineering process. The key feature of such tools are predefined user interfaces, the degree of formalization, and the way of how the knowledge is organized. However, often it appears, that for a given project, considering its contributors, domain and goal, a more appropriate solution might be imaginable, but not yet existing. Customizable and extensible (non wiki-based) tools for building knowledge-based systems are available today (e.g., Protege [1]). Another approach is building a customized knowledge acquisition tool from scratch. However, both approaches do not allow for evolutionary incremental specification and implementation of the tool in parallel with the (beginning) knowledge acquisition activities, but require the specification and implementation in advance. The specification of a knowledge engineering tool in advance bears another challenge: At the beginning of the cooperation of knowledge engineers and domain experts a sound specification is ambitious and risky. As the knowledge engineers initially are not familiar with the domain and the contributors (experience level, number, availability) to be able to define the best possible knowledge formalization method. We propose an incremental and agile approach, that allows for the smooth but immediate startup of knowledge formation and breaks down the constraints and entry barriers to a minimum. The (semantic) wiki-based approach demands from the contributors at the beginning only the capability of mastering the basic wiki workflow, which is browsing and modifying wiki pages. Its only

constraint is, that the knowledge can be entered or imported as wiki content and partitioned/organized into wiki pages. We argue, that this method retains a high level of flexibility being able to support a large number of requirements. However, the gap between a wiki filled with domain knowledge and an executable knowledge-based system, using a formal knowledge representation, is still large. In this paper, we discuss how this gap (emerging on conceptual and on technical level) can be bridged in an agile and incremental manner with reduced delay of the knowledge acquisition phase, and at moderate (software) engineering costs. The knowledge engineering tasks, we focus on, are the development of decision-support systems where solutions, based on entered (formal) problem descriptions in sessions, are proposed. We demonstrate the meta-engineering process for this task by sketching its implementation by several case studies.

The rest of the paper is organized as follows: In Section 2 the meta-engineering process is explained in more detail, considering the conceptual and the technical level. Further, we discuss in Section 3 how the Semantic Wiki KnowWE supports the meta-engineering idea on the technical level. Demonstrating the applicability of the proposed approach, we report on experiences made in different projects in Section 4. We conclude the paper by giving a summary and pointing out directions for future work.

2 The Meta-Engineering Process

The meta-engineering process proposes to model a knowledge acquisition process, that is free from any knowledge engineering tool or knowledge representation at the beginning. The initial phase tries to envision the optimal knowledge formalization environment for the project without regarding any technical constraints. The result is then developed towards a project tailored specification of a knowledge engineering tool. We argue that a wiki poses very low constraints, only demanding that the knowledge can be defined in some textual form, and because of this forms suitable basis for this approach. Thus, the question in the initial phase is, how the knowledge can be entered in a wiki in a *formalizable* way, regarding domain, contributors, startup knowledge, and goal. We call the result of this process the *wiki-based formalization architecture* optimizing criterias such as understandability, maintainability, and acquisition efficiency - yet disregarding any technical constraints. Figure 1 shows the cooperative phases of the meta-engineering process.

Fig. 1. The cooperative phases of the wiki-based meta-engineering process.

At first, in the experimental phase a small toy prototype is implemented using formalization methods (markup, reasoner) already provided by the tool. Even though, the available markups may be not optimal for this task, this phase gives the domain specialists an impression of wiki-based knowledge acquisition. Then, in the design phase possible markups and partitionings for the knowledge are developed in an iterative manner, forming the wiki-based formalization architecture. Small parts of the domain knowledge are entered in the wiki by using the currently specified architecture. Although, the tool cannot compile and process the knowledge at this point, discussing these prototypical knowledge artifacts can catalyze the exchange of expertise. Knowledge engineers obtain an impression of the knowledge that needs to be formalized, and domain specialists experience the general idea of wiki-based knowledge formalization. This allows for a better estimation of the formalization architecture criterias *understandability*, *maintainability*, and *acquisition efficiency*. That way, the formalization architecture can be revised and refined iteratively in joint discussion sessions. Due to the flexibility of the wiki approach these phases can easily overlap resulting in an agile specification and development process of the tool. The process finally leads to the main phase when design and implementation activities get finished, featuring a thoroughly tailored knowledge engineering environment.

2.1 Conceptual Level: Bridging the Gap of Expertise

As already stated in the introduction, it is often difficult to completely specify the most appropriate acquisition method and tool at project startup. Often, either the knowledge engineer or the domain specialist are not familiar enough with the other discipline respectively at the beginning. The wiki-based meta-engineering approach tries to overcome this gap of expertise by the two cooperative phases of experimentation and design. The *wiki-based formalization architecture*, forming the result of these phases, contains the following aspects:

1. **Identification and representation of important domain concepts:** This task defines how the domain concepts are represented in the wiki. Informal "support knowledge" (e.g., textual descriptions, images, links) is added to the concepts, describing the concept, defining a common grounding of their meaning, and documenting its role in the formal model of the domain. When possible, the support knowledge is streamed into the wiki by reusing legacy documents of the project context.
2. **The distribution of the formal knowledge:** The knowledge formalization architecture defines how the formal knowledge is organized in the wiki. The derivation knowledge typically connects the input concepts (findings of the problem description) with the requested output concepts. In general, the derivation knowledge is distributed according to a domain-dependent partitioning and attached to the wiki pages of the most related concepts. However, there is not yet a canonical receipt to select or create the optimal knowledge distribution for a given project in one step.

3. **Definition of the markup:** When defining the appropriate markup general design principles for domain specific languages (DSL) should be adhered to. Spinellis [2] emphasizes, for example, to include the domain experts closely into the design process. Karsai et al. [3] report about design guidelines like the reuse of existing language definitions and type systems or limitation of the number of language elements. In the context of wikis, also the intuitive and seamless integration with the informal support knowledge has to be considered. Hence, the documents are not exclusively created by using the DSL, which is in this case only forming the markup for fragments of the overall wiki content. Knowledge markups in wikis can be designed for use at different (syntactical) granularities: For example, large sections like table- or list-based markups or small relation atoms, that can be distributed as single items within the (informal) page content, are possible [4]. The first allows for a more comprehensive view on the formal knowledge fragments in an aggregated form. The latter in general allows for a better integration with the informal knowledge each relation being injected at the most suitable location in the text. In this case, additional views should be generated from the spread relations to provide concise overviews, for example by using inline query mechanisms.

In cooperative sessions the knowledge engineers together with the domain specialists try out different possibilities for the described aspects. For each idea some demo wiki pages, covering a very small part of the domain, are created, disregarding that the knowledge is not (yet) processed by the wiki.

When this iterative design process has lead to a promising result the implementation phase of the meta-engineering process begins, aiming at modifying the tool to be able to parse and compile the knowledge, according to the designed formalization architecture. At this point, the already inserted knowledge of the demo pages on the one hand can be kept, forming the first part of the knowledge base, and on the other hand serves as a specification for the markup.

2.2 Technical Level: Bridging the Gap of Tool Support

The design phase on the conceptual level identifies a specification of an appropriate (wiki-based) knowledge formalization architecture, for which in most cases no tool support is existing at that point. The gap between a standard wiki or even a standard semantic wiki to the envisioned tool in most cases is still large, for example if the support of production rules entered by a custom markup should be supported. However, the general process of parsing and compilation the knowledge in a wiki-based workflow is always similar. Figure 2 shows an outline of the wiki-based knowledge formalization process chain from the knowledge source (domain specialists or legacy documents) on the left to the productive running system on the right. There are four main steps involved, which are discussed in the following.

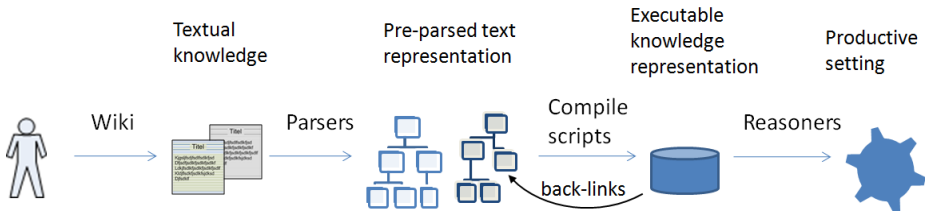

Fig. 2. The wiki-based knowledge engineering process chain.

1. **A wiki to create textual knowledge:** This is the essential part of a wiki application. The wiki interface is used to create textual documents. In general, any standard wiki engine can be used to accomplish this task.
2. **Parsers to create a pre-parsed representation of the textual knowledge:** To create formalized knowledge from text documents, markup needs to be defined. For the specific markup the corresponding parser components are integrated into the system. They create a (concrete) syntax tree of the wiki pages (also containing large nodes of free text). In this parse tree the structure of the formal relations, i.e., references on formal concepts and their relations, are contained.
3. **Compilation scripts to create executable knowledge:** The compile scripts transform the pre-parsed representation of the markup into an executable knowledge format, that can be interpreted by the reasoners. The compile scripts need to be defined with respect to the markup and its syntax tree representation and the target data structure defined by the intended reasoning engine.
4. **Reasoners to test the knowledge base:** Any reasoner that can solve the intended reasoning task of the application can be integrated. For the evolutionary development, testing of the knowledge base is necessary. Hence, components for the execution of the reasoner with the knowledge base need to be provided.

In general, the steps of parsing (2) and compilation (3) could be considered as one step in the process chain. However, separating them into two steps by the use of some structured text-representation has important advantages: Back-links from the formal knowledge artifacts to the corresponding, original text entities become possible. This allows to identify for each formal relation the exact location in the text that it was generated from. One can make use of this for the implementation of important tasks:

- **Explanation:** Explanation components presenting the text slices, that are responsible for the current reasoning result, can be created.
- **Validation:** For many knowledge representations and reasoners validation methods exist, detecting deficiencies like redundant or inconsistent knowledge.

Without the back-links the text location of the corresponding knowledge artifacts can not be identified to correct or tidy the wiki content, being the source

of the compilation process. In general, these two techniques—explanation and validation—are truly necessary to build up large well-formed knowledge bases using an agile methodology. Further, algorithms for refactoring of the knowledge heavily benefit from a pre-parsed text representation, and when exchanging the target reasoning engine only the compile scripts need to be modified and the parsing components remain untouched.

To help bridging the technical gap spanned by the designed formalization architecture and some existing tool, we propose the design of a framework. It needs to allow for the easy integration of missing elements and to provide a library of reusable components to fill the gaps in the formalization chain.

3 Meta-Engineering with KnowWE

KnowWE [5] has been designed to support the implementation of tailored formalization architectures in a process chain like sketched in 2. It provides a library of components that can be reused in the contexts of different projects and allows for the definition and integration of custom components at low implementation costs. KnowWE connects several free and open source software components to provide the knowledge formalization capabilities sketched in Figure 2. As basic wiki engine JSPWiki[1] is used. We integrated the reasoning engines *OWLIM*[2] for RDF reasoning and *d3web*[3] for diagnostic reasoning. To extend the system with additional components (e.g., parser, compile scripts, renderers,...), we provide a flexible plugin mechanism based on *JPF (Java Plugin Framework*[4]*)*. Besides the interconnection of these components forming a semantic wiki with problem-solving capabilities, the major technical contribution of KnowWE is the generic typed data-structure for the pre-parsed representation of the textual content as shown in Figure 2, called *KDOM* (Knowledge Document Object Model). The wiki documents are parsed according to the extensible *KDOM schema*, where all allowed textual entities are defined in a declarative way. A detailed explanation of the parsing algorithm creating the parse-tree of a document using a specified KDOM schema can be found in [6].

3.1 Parsing

The KnowWE core library contains components to configure a project specific KDOM schema. While parser components for common useful syntactical structures such as tables, bullet lists, dash-trees, or XML are provided by the system, for domain specific languages own parser components need to be defined. Figure 3 shows different applications of the dash-tree markup as decision tree, concept hierarchy and property hierarchy.

[1] http://www.jspwiki.org
[2] http://www.ontotext.com/owlim/
[3] http://www.d3web.de
[4] http://jpf.sourceforge.net/

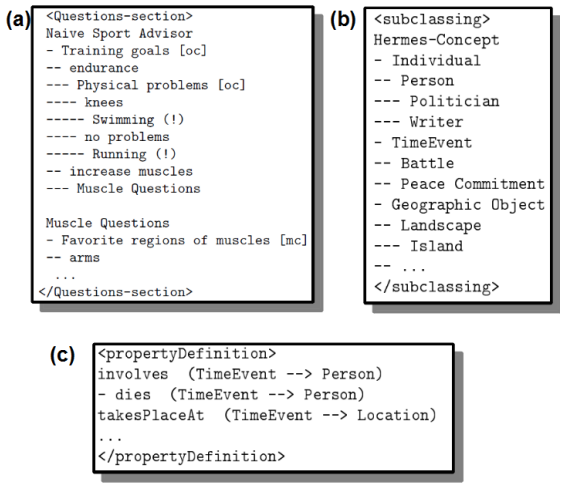

Fig. 3. Different applications of dash-tree based markups: (a) as decision tree in a sports advisor knowledge base; (b) as concept hierarchy in the historical domain; (c) as property hierarchy with domain and range definition

Customized reuse of predefined markup is possible by small modifications in the dash-tree KDOM schema components. Figure 4 shows the basic KDOM schema of the dash tree markup. The blue nodes represent KDOM types from the KnowWE core library, provided with the corresponding parsing component (i.e., regular expressions). Only small modifications at the dash-tree leaf of the schema are necessary to enable specific parsing and compilation tasks using dash-trees.

Recently, a bridge to the UIMA Framework[5], which is an open source framework for unstructured/semi-structured information processing, was integrated. Currently, experiments extracting formal knowledge from legacy documents, using different of the large number of UIMA analysis engines available, are run. The configuration of tailored information extraction components as a meta-engineering task, alternatively to the development of markup, aims at semi-automation of the knowledge acquisition process.

3.2 Compilation and Reasoning

Compile scripts can be attached to the KDOM schema being executed automatically after the corresponding subtree has been instantiated by the parsing process of a page. They walk the KDOM parse-tree and instantiate the executable knowledge in the corresponding repository, depending on the targeted reasoning engine. By the use of the unique IDs of the nodes of the KDOM nodes, the back-links described in Section 2 can be created.

[5] http://uima-framework.sourceforge.net/

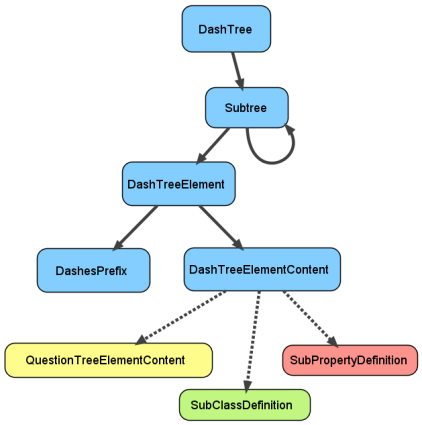

Fig. 4. A KDOM schema defining dash-tree syntax. Parsing and compilation of the leaf contents can be customized.

A general include mechanism, which is independent of markup or target representation, allows for specific compiling tasks. By using the include mechanism, any composition of knowledge fragments from various pages can be assembled to one wiki page for testing, generation of views, creation of variants, or export of knowledge bases. The KnowWE core currently integrates (swift-) OWLIM for RDF reasoning. Further, we integrated the d3web engine for diagnostic problem-solving together with basic markups [4] for knowledge formalization in d3web. The integration of the open source rule engine *Drools*[6] is currently in progress. For many applications these reasoners should provide sufficient possibilities to build at least a small prototypical executable knowledge base for demonstration supporting the initial cooperative design phase. The potential later integration of an optimized reasoning engine (e.g., that better scales on the specific reasoning task) only demands to modify the compile scripts and to provide an endpoint for the execution of the engine for testing purposes.

3.3 Refactoring

Meta-engineering proposes an evolutionary approach with respect to the (wiki-based) knowledge formalization architecture implying experimentation with different partitionings of the knowledge over the wiki pages and different markups. Therefore, for transferring content from one page to another or transforming one markup to another, automated support is crucial to prevent repetitions in the knowledge acquisition activities. To allow for efficient scripting of these transformations, we integrated a Groovy[7] endpoint into the wiki, accessible only to project admins. It allows to create and execute transformation scripts on the

[6] http://labs.jboss.com/drools
[7] http://groovy.codehaus.org

wiki content. The Groovy scripts makes use of a predefined API to access, scan and modify the KDOM data-structure. Figure 5 shows a refactoring script embedded in the wiki. At the top of the wiki page a GUI widget for the execution of the refactoring script is shown. Underneath, a Groovy script for renaming an object is located. As these refactoring script operations are complex and dangerous, they should be performed in offline mode (i.e., blocking wiki access for the general users at execution time).

Fig. 5. A refactoring script in the wiki written in Groovy, which renames an object globally.

4 Case Studies

We are currently evaluating the meta-engineering approach within several industrial and academic projects using the KnowWE system. They are addressing a wide range of different domains like biology, chemistry, medicine, history, and technical devices. In some projects, the customizations only make small changes to the existing features while in others large components are created. In the following, we briefly introduce the projects and explain the employed meta-engineering methods.

4.1 Clinical Process Knowledge with CliWE

In another recent project, KnowWE is extended by diagnostic workflow knowledge in the context of the CliWE project[8]. This project considers the development of a medical diagnosis system for a closed-loop device. Documents describing the domain knowledge already exist and these are imported into the wiki as textual and tabular information about the domain. The particular wiki articles are focusing on special aspects of the diagnosis task, for example the assessment of the current patient's state. At predefined milestones the knowledge in the wiki is exported into a single knowledge base in order to evaluate the development on real-time devices.

In general, the core knowledge formalization methods from the d3web-plugin are used. However, to implement closed-loop systems, the need for an additional knowledge representation and reasoning support was identified at an early stage of the project. Thus, a flowchart-like visual knowledge representation has been designed. The flowcharts can be edited in the wiki using the integrated visual flowchart editor *DiaFlux*. To allow for modularization the flowcharts can be organized hierarchically. A (sub-) flowchart can be included into another flowchart as one (box-) component by defining and connecting the input/output interface. Due to this hierarchical organization, partitioning of the different aspects of the domain knowledge over the wiki, is possible. A first prototype of this extension is reported in Hatko et al. [9].

4.2 Fault Diagnosis for Special Purpose Vehicles

Another project considers the development of a diagnostic system for special purpose vehicles. The goal is to reduce the repair time and costs of the vehicles by determining the faulty element by a cost-minimal sequence of (potentially tedious) examinations. The system is build based on existing construction plans and heuristic knowledge of experienced mechanics. After an analysis phase the wiki formalization architecture has been defined. It contains one structural model of the vehicle, one state model of the current diagnosis session and for each technical subcomponent fault causes and malfunctions. For each of this knowledge base components own markup has been defined allowing logical distribution of the knowledge base over different wiki pages and seamless integration with support knowledge, such as technical documents and construction plans. The knowledge will be compiled into set-covering knowledge containing also the cost values for any examination at some given state for a sophisticated interview strategy calculated by an additional problem-solver. This wiki-based formalization architecture has been defined after an initial phase where one Excel-based approach and one wiki based approach using existing markup have been evaluated. Including a initial phase with iterative cooperative sessions and finally the technical implementation of the defined formalization architecture, the project shows a successful application of the Meta-Engineering approach.

[8] CliWE (Clinical Wiki Environment) is funded by Drägerwerk, Germany and runs from 2009-2012.

4.3 WISEC

The WISEC (**W**iki for **I**dentified **S**ubstances of **E**cological **C**oncern) project[9] investigates the management and detection of substances with respect to its bio-chemical characteristics. Here, substances of very high concern (SVHC) under environmental protection considerations are investigated and managed using the multi-modal approach of a Semantic Wiki: The knowledge about each substance is represented by an wiki article containing informal descriptions of the substance and its relations to external sources (via semantic annotations). The overall knowledge base also integrates already known lists of critical substances and explicit domain knowledge of a specialists combining the particular characteristics of the criticality of substances.

Tailored markups were created to capture the relation of the substances to already known critical substance lists. Thus, a list of critical substances in the wiki is still human-readable, but is also automatically compiled as a collection of ontological properties relating the substance concepts with the list concept. Furthermore, special properties (such as different toxic contributions) are also parsed as formal properties of the list concepts. Due to the explicit representation of the relational knowledge in OWL, different properties of substances can be queried over the wiki knowledge base.

4.4 Medical Decision-Support with CareMate

The *CareMate* system is a consultation system for 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 focusing 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.

For the CareMate project, the core entities of the formalization architecture 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, and the corresponding derivation knowledge is defined using the knowledge formalization pattern *heuristic decision tree* [7]. In Figure 6 the wiki article of the cardinal symptom stomach pain ("Bauchschmerzen") is shown.

[9] in cooperation with the Federal Environment Agency (UBA), Germany

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.

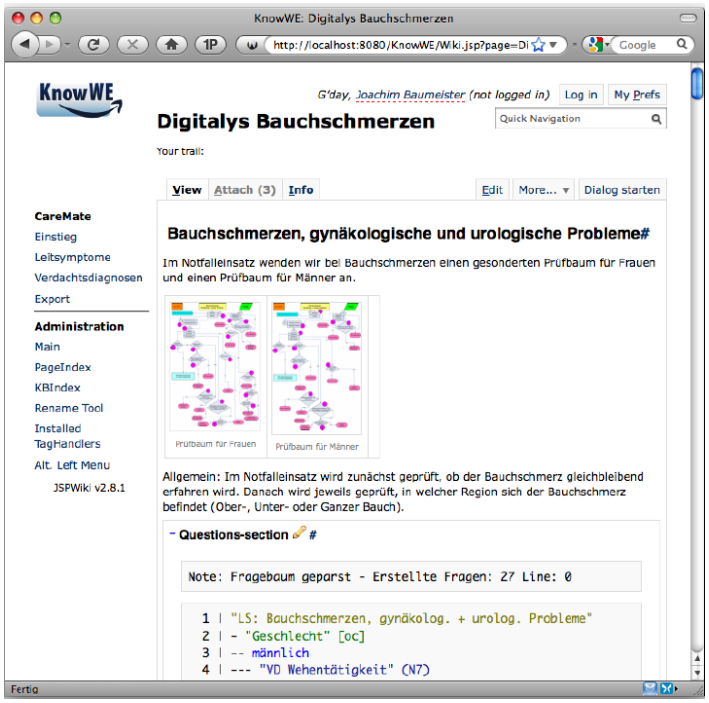

Fig. 6. The wiki page about the cardinal symptom "Bauchschmerzen" (stomach pain). (Screenshot in German language)

The CareMate system is commercially sold by the company Digitalys[10] as part of an equipment kit for medical rescue trucks.

4.5 Biodiversity with BIOLOG

The BIOLOG Europe project[11] aims at integrating socio-economic and landscape ecological research to study the effects of environmental change on managed ecosystems. To make the results of the research accessible for domain

[10] http://www.digitalys.de
[11] www.biolog-europe.org

specialists as well as diverse people with a different background, they decided to build a knowledge system (decision-support system). BIOLOG Wissen (BIOLOG Knowledge) is based on KnowWE and serves as a web-based application for the collaborative construction and use of the decision-support system in the domain of landscape diversity. It aims to integrate knowledge on causal dependencies of stakeholders, relevant statistical data, and multimedia content. In addition to the core formalization methods of the d3web-plugin, we introduced some domain specific features: One major challenge for the researchers in this domain is to find related work about similar studies. For this reason the research community (of ecology) has defined an extensive XML-schema for the description of meta-data about ecological work called *EML* (Ecological Meta-Data Language[12]). We defined a sub-language of EML which is suited to support capturing the relevant meta-data in this project. The research results and examinations of the different BIOLOG sub-projects are provided in EML and are entered into the wiki. Then, the EML data sets are visualized and can be accessed through a (semantic) search interface. Figure 7 shows the BIOLOG wiki depicting (part of) the visualization of an EML data set that describes work about *perception and appreciation of biodiversity* ("Wahrnehmung und Wertschätzung Biodiversität").

Fig. 7. Visualization of the EML data set about *perception and appreciation of biodiversity* (Screenshot in German language).

As the domain specialists are used to model concept hierarchies in the mind-mapping tool *FreeMap*, we integrated support for the FreeMap XML format in the wiki. Thus, a hierarchy can be created or modified externally with FreeMap and then pasted into the wiki to be translated into OWL-concepts.

[12] http://knb.ecoinformatics.org/software/eml/eml-2.0.1/index.html

To manage the publications of the project efficiently, we integrated support of BibTeX data. The wiki serves as a bibliography data base for the publications that have been created within the project scope.

4.6 Ancient History with HermesWiki

The HermesWiki [8] is a semantic wiki in the domain of Ancient Greek History. It is developed in cooperation the Department of Ancient Greek History of the University of Würzburg, Germany. Even though the HermesWiki does not develop a decision support system the meta-engineering approach has been applied successfully on both levels — conceptual and technical.

- **Conceptual Level** As the project at the beginning used a regular (non-semantic) wiki, at first the knowledge acquisition process clearly was free from any constraints due to knowledge formalization. After it was clear how the domain experts structured the content in the wiki in a natural way, we began to integrate formalization mechanisms tailored to the content already given in the wiki and to the workflow of the contributors. For example, we discovered that often multiple (related) time events were described on one page and defined a markup allowing to formalize a text paragraph as a time event by adding a title, a time stamp, and references to (historical) sources.
- **Technical Level** The HermesWiki has been implemented as a plugin for KnowWE reusing as much of the provided core components as possible. While some standard markups (e.g., annotation of pages being instance of some class) could be reused others had to be added (e.g., for time events or locations). Further, the dash-tree markup could be reused in different ways to define hierarchical structures. While the dash-tree parser from the core is used to parse the tree structure, the plugin only needs to specify how to process the (dash-tree) nodes during the compile process with respect to their father or children nodes.

Further details about the formalization methods of the HermesWiki can be found in Reutelshoefer et al. [8]. There, also the use cases for the formalized knowledge in the context of e-learning are described.

5 Conclusion

In this paper we introduced the idea of meta-engineering as an alternative approach to using 'out of the box' systems and building new ones from scratch. We motivated how the meta-engineering approach can help to bridge the two worlds of knowledge engineers and domain specialists and catalyzes the creation of a project tailored knowledge acquisition tool. Semantic Wikis are the appropriate technical platform to implement that approach, as a wiki builds a flexible basis and is customizable to a wide range of knowledge acquisition scenarios. They further allow for initial design phases, where specification and knowledge

acquisition can run in parallel. We discussed how the Semantic Wiki KnowWE supports the meta-engineering idea and we reported about several projects from different domains where the method proved to be helpful for the overall knowledge acquisition efforts. The introduced meta-engineering approach in principle also can be applied by the use of other Semantic Wiki systems, which are also designed with component based extension mechanisms, like for example Semantic MediaWiki [10] and KiWi [11]. However, no other systems known provide components for explicitly building intelligent decision-support systems. We still need to gather more experiences on how to determine the most appropriate wiki-based formalization architecture for a given project. Further, we will improve the technical infrastructure of KnowWE to allow meta-engineering to be applied with even lower implementation efforts.

References

1. Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., Musen, M.A.: Creating Semantic Web contents with Protege-2000. IEEE Intelligent Systems **16**(2) (2001) 60–71
2. Spinellis, D.: Notable Design Patterns for Domain Specific Languages. Journal of Systems and Software **56**(1) (February 2001) 91–99
3. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schneider, M., Völkel, S.: Design guidelines for domain specific languages. In Rossi, M., Sprinkle, J., Gray, J., Tolvanen, J.P., eds.: Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM09). (2009) 7–13
4. Baumeister, J., Reutelshoefer, J., Puppe, F.: Markups for Knowledge Wikis. In: SAAKM'07: Proceedings of the Semantic Authoring, Annotation and Knowledge Markup Workshop, Whistler, Canada (2007) 7–14
5. Baumeister, J., Reutelshoefer, J., Puppe, F.: KnowWE: A Semantic Wiki for Knowledge Engineering. Applied Intelligence (2010)
6. Reutelshoefer, J., Baumeister, J., Puppe, F.: A Data Structure for the Refactoring of Multimodal Knowledge. In: 5th Workshop on Knowledge Engineering and Software Engineering (KESE). CEUR workshop preceedings, Paderborn, CEUR-WS.org (September 2009)
7. Puppe, F.: Knowledge Formalization Patterns. In: Proceedings of PKAW 2000, Sydney Australia. (2000)
8. Reutelshoefer, J., Lemmerich, F., Baumeister, J., Wintjes, J., Haas, L.: Taking OWL to Athens – Semantic Web technology takes Ancient Greek history to students. In: ESWC'10: Proceedings of the 7th Extended Semantic Web Conference, Springer (2010)
9. Hatko, R., Belli, V., Baumeister, J.: Modelling Diagnostic Flows in Wikis. In: LWA-2009 (Special Track on Knowledge Management), Darmstadt (2009)
10. Krötzsch, M., Vrandecic, D., Völkel, M.: Semantic MediaWiki. In: The Semantic Web - ISWC 2006. Volume 4273 of Lecture Notes in Computer Science., Heidelberg, DE, Springer (2006) 935–942
11. Schaffert, S., Eder, J., Grünwald, S., Kurz, T., Radulescu, M.: KiWi - A Platform for Semantic Social Software (Demonstration). In: ESWC'09: The Semantic Web: Research and Applications, Proceedings of the 6th European Semantic Web Conference, Heraklion, Greece (June 2009) 888–892