A Meta-Engineering Approach for Customized Document-centered Knowledge Acquisition

Jochen Reutelshoefer, Joachim Baumeister and Frank Puppe

Department of Intelligent Systems
University of Würzburg
Am Hubland
97074 Würzburg
reutelshoefer@informatik.uni-wuerzburg.de
joachim.baumeister@denkbares.com
puppe@informatik.uni-wuerzburg.de

Abstract: Enabling domain specialists to formalize domain knowledge using knowledge acquisition tools, also known as direct knowledge acquisition, has been a challenge in artificial intelligence research for decades. Building custom-tailored knowledge acquisition tools proved to be a convenient method to empower domain specialists to contribute to knowledge bases directly. Engineering on the meta level, i.e. building tools that allow to build customized knowledge acquisition tools, is the strategy to reduce implementation costs for the customized tools. In this paper, we present an approach to build customized knowledge acquisition environments for document-centered knowledge base authoring. This editing paradigm, being an alternative to the use of graphical user interfaces, is well suited for the customization approach allowing for knowledge acquisition at low effort for both, the domain specialists and the knowledge engineers. We present a meta-engineering process coordinating the customization efforts and a metatool for the system KnowWE enabling efficient meta-engineering. To illustrate the overall approach we describe a real-world case study in e-learning in the domain of Ancient History.

1 Introduction

Modeling the knowledge of a specific domain as a computer interpretable knowledge base to solve knowledge-intense decision problems (e.g., diagnosis tasks) has been a major research topic in artificial intelligence since decades. The dilemma of this task is that the specialists of the respective domain are usually not experts in modeling domain knowledge by means of formal knowledge representations (e.g., rules, logics). On the other hand, the people qualified in these modeling skills are typically not familiar with the domain. For manual knowledge acquisition (KA) for knowledge-based systems, we can distinguish the two major strategies [PG92], *direct KA* and *indirect KA*. While in direct KA the domain experts are creating (at least major parts of) the knowledge base autonomously, the indirect approach implies that knowledge engineers obtain the knowledge from the domain experts (e.g., in interview sessions) and then implement it into the knowledge base. As the latter implies high personnel expenditure also considering long term maintenance, direct KA appears to be more promising if it can be applied efficiently. There, the key problem is that domain specialists are not trained in knowledge engineering or in using knowledge acquisition/modeling tools. Musen et. al at first established the idea of building tools tailored to the mental models of the domain specialists to overcome this problem:

"The conceptual model thus forms the basis of a language with which both the tool and the tool's user can describe the contents of a knowledge base." [Mus88]

Consequently, if the tool complies to a conceptual model shared by the domain specialists, using the tool should be easy for them. This involves the specification of such conceptual model and a complying user interface/tool. While this approach lowers the barriers for domain specialists to contribute and makes direct knowledge acquisition more efficient, it implies the costly implementation of the respective custom tailored tools for each project/domain. A strategy to reduce these development costs is developing (domain independent) meta-level tools [EM93], that can generate custom tailored knowledge acquisition tools from high-level descriptions of the domain and the knowledge acquisition task. In the following we refer to the specification and implementation task of a custom-tailored knowledge acquisition tool, possibly including the use of a metatool, as *meta (knowledge) engineering*. Until now, meta engineering for knowledge acquisition tools was focused on graphical user interfaces. However, there is another editing paradigm that can be a beneficial alternative to the use of graphical user interfaces, which we call the *document-centered knowledge authoring* approach. There, the user interacts with the knowledge base indirectly by editing documents, e.g., by using some basic text editing tool. Segments complying to some predefined syntax are automatically processed and compiled to the formal model of the knowledge base. In this paper, we apply the meta-engineering approach, i.e., the creation of custom tailored domain specific user interfaces, to document-centered authoring. Despite of the formation of a highly customized tool allowing for direct knowledge acquisition, this combination brings along further advantages. First, it allows for knowledge acquisition activities and tool specification/customization running in parallel. Furthermore, it enables participation of domain specialists at a diverse range of technical skills with low contribution barriers. At last, we show that metatools can also be used effectively in this context to reduce implementation costs of tool customization.

The rest of the paper is structured as follows: In Section 2, we explain the document-centered knowledge base authoring approach and its benefits in more detail. After that, the meta-engineering process, applying a stepwise customization of the document-centered authoring environment according to the current project, is discussed in Section 3. In Section 4, we present a real world case study to demonstrate the approach in practice. We introduce an implementation of the document-centered authoring environment *KnowWE* and a corresponding metatool, that allows for domain specific customization at low engineering costs, in Section 5. In Section 6, the approach is compared to related work before we conclude with a summary and outlook in Section 7.

2 Document-centered Knowledge Base Authoring

In this section, we explain the document-centered authoring approach for knowledge bases in more detail and point out some of its advantages including its potential for customization. Most authoring environments for creating formal knowledge bases offer graphical user interfaces. There the authors can use input forms, which are organized by menus or tabs, to enter the knowledge (e.g., Protégé[1] [GMF$^+$03], CLASSIKA [PG92]). Document-centered knowledge base authoring is an alternative authoring paradigm, providing different benefits and challenges with respect to customization. There, the authoring environment provides access to a set of documents, that can be modified and extended by the users, employing some basic text-editing interface. The user can freely structure and edit these documents basically without being constrained by the tool. To actually create components of the formal knowledge base, the user has to comply to a formal syntax provided by the authoring environment. Statements complying to this syntax are then translated to the knowledge base as shown in Figure 1. The process of compiling the documents to the (interpretable format of) the knowledge base decouples the user from this machine readable representation of the knowledge. In this way, the documents, that can be structured according to the users needs, are forming a kind of human-oriented representation layer of the knowledge. A document-centered authoring environment manages the documents in a centralized repository and processes all document modifications performed by the users by updating the knowledge base accordingly. Figure 2 shows an excerpt of an example document from a car-fault diagnosis knowledge base using rules. On the top (1) one can see some informal text content describing the domain. Below (2) a rule syntax is used within the document. This rule syntax is compiled and the corresponding executable rules are inserted into the knowledge base after each document modification. The updated version of the knowledge base is then ready for testing instantly. In the following we refer to such kind of syntax, together with some translation instruction transforming it to the knowledge base repository, a *(knowledge) markup language*. This approach in general can be applied to any kinds of computer interpretable knowledge representations, as for example various rule-based or logics-based formalisms. This authoring paradigm is similar to how software is typically developed since decades. As experienced in software engineering, this method of authoring strongly depends on good user assistance in terms of feedback and authoring assistance. Those techniques have successfully been evolved within common programming environments in recent years and similarly can be applied to knowledge authoring.
Being dominant in software engineering, document-centered authoring has not yet been

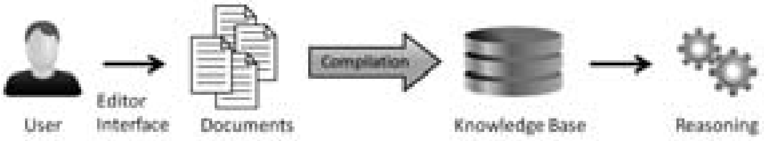
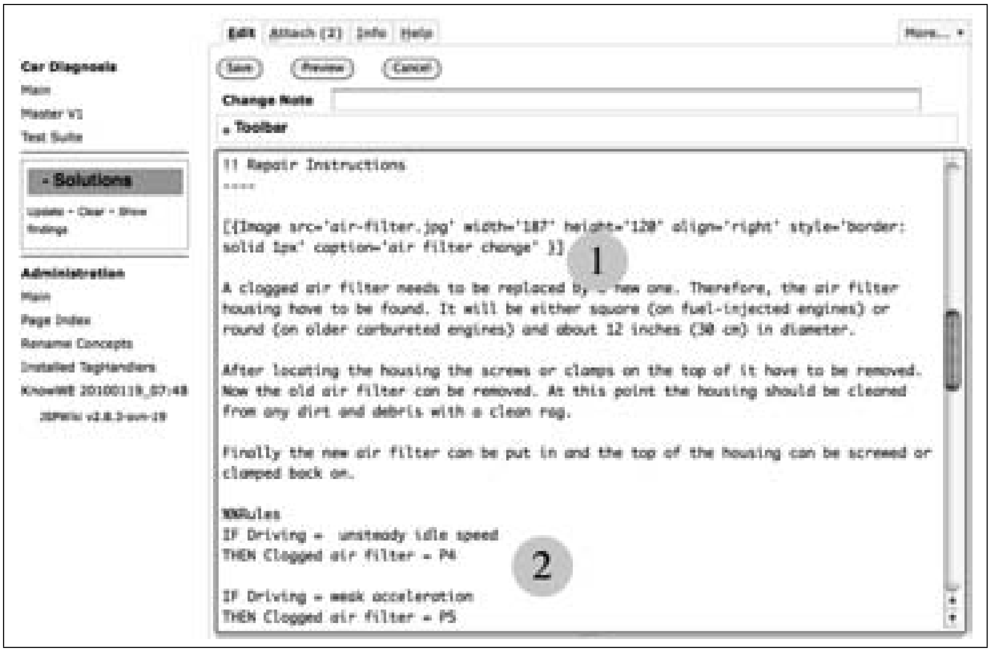


Figure 1: Structure of the document-centered knowledge authoring paradigm.

[1]http://protege.standford.edu



```
!! Repair Instructions
----

[{Image src='air-filter.jpg' width='187' height='120' align='right' style='border:
solid 1px' caption='air filter change' }]

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

After locating the housing the screws or clamps on the top of it have to be removed.
Now the old air filter can be removed. At this point the housing should be cleaned
from any dirt and debris with a clean rag.

Finally the new air filter can be put in and the top of the housing can be screwed or
clamped back on.

%%Rules
IF Driving =  unsteady idle speed
THEN Clogged air filter = P4

IF Driving = weak acceleration
THEN Clogged air filter = P5
```

Figure 2: An excerpt of an example document from a car-fault diagnosis knowledge base using rules.

applied in large scale for modeling knowledge bases. Compared to software engineering, here the challenge is, that knowledge engineering projects usually bring together people with different backgrounds and strongly differing expertise with respect to the domain and knowledge engineering. While in software engineering the source documents usually are only edited by software engineers, the direct knowledge acquisition approach implies that also domain specialists, often with low technical backgrounds, step into the author role. In the following, we describe some advantages of using this authoring paradigm for knowledge base development that we experienced in several projects.

Low Barriers for Basic Contributions The level of the technical skills of the participating users typically is rather diverse in knowledge engineering projects. Therefore, it is important to enable low barrier contribution according to the specific level of expertise of a participant. Editing text documents is a rather simple editing paradigm, when compared with some complex menu- and form-based user interfaces existing. It allows for basic contributions (e.g., adding informal descriptions, proof reading) without any training of a new tool or further expertise as most people are already used to read and maintain content as electronic documents. Being used to these kind of simple contributions without difficulty, contributors often feel encouraged to explore more complex tasks.

Example-based Authoring While contributions on informal parts of the documents are possible without training, the actual knowledge base can only be modified by using the knowledge markup language in a proper way. The idea of example-based learning proposes, that initially knowledge markup statements are inserted into the document base (either modeling initial parts of the knowledge base or some toy example). If a user can comprehend the meaning of these statements, he can easily adopt it for himself to express other knowledge using simple copy/paste&modify. Often only the entity names need to be exchanged to create new valid knowledge.

Incremental Formalization The process of *incremental formalization* starts with the insertion of informal content describing the domain, such as text and figures. This content is either created by domain specialists or adopted from documents often already existing in the domain context. At first, it serves as a startup for the formalization process and later it forms the documentation and context of the knowledge base components. The incremental formalization process proceeds with the identification of those content parts, that need to be formalized to form the intended executable knowledge base. After that, a tentative formalization is made, that is, the selected content is transformed towards the knowledge markup language. This initial, potentially erroneous or incomplete, formalization can then be refined gradually. These distinct steps require different degrees of expertise in the domain, in knowledge engineering, and usage of the respective acquisition tool. These different kinds of competencies often are distributed heterogeneously. Therefore, a decomposition of the formalization tasks into distinct steps, possibly involving different persons on different steps, simplifies the accomplishment of the formalization task. Hence, the incremental formalization workflow helps different participants to be able to contribute according to their respective capabilities.

Quality Management Quality management is a crucial and challenging task in the knowledge base development and maintenance process. Also in the domain of software engineering the aspect of quality management has been studied for many years. A very successful set of practices that was established within the last decade, especially in the context of agile software development [BA04, Mar09], is called *Continuous Integration* (CI). According to Fowler[2] the main requirements for CI are the use of a code repository, automated building, automated tests, frequent and timely integration of changes, and easy access to the latest builds. The main benefits of CI are, that always a valid version of the system is available for deployment in a productive setting. Further, problems emerging by changes are recognized very early, making debugging easy and reducing risks of complex change operations. The knowledge base authoring approach described in this paper allows for straight forward application of CI, as documents can easily be put under version control in a centralized repository. Hence, by using CI-based development as known from software engineering, it is possible to continuously guarantee quality and transparency.

Freedom of Structuring The partition of the content in documents can freely be chosen (including the names of the document). Within one document the order of the paragraphs and knowledge markups statements is free to the user. Informal support knowledge, e.g.,

[2]http://martinfowler.com/articles/continuousIntegration.html

comments and figures, can be inserted at any place and in any style. Further, the documents can freely be interlinked with others making interrelations of content parts explicit and improve navigation. The freedom of structuring allows the documents to evolve a memorable and comprehensible structure, becoming familiar to the authors. In software engineering, it is known that while working on programming code, the amount of time spent on reading compared to the amount of time spent on actually editing is more than ten to one [Mar09] and therefore readability (and comprehensibility) is a focal point. In document-centered knowledge acquisition complex digital artifacts are created in a similar way demanding similar cognitive challenges to the authors. Therefore, we also focus on the aspect of readability in this approach and describe in more detail the degrees of freedom of the structuring of a document-centered knowledge base affecting that aspect:

1. **Support Knowledge:** This aspect defines what kind of (informal) domain knowledge is contained in the document corpus and how it is organized. The overall content should cover all domain knowledge that is relevant for the intended knowledge base. This information needs to be partitioned into (interlinked) documents. Usually this is each document treating one sub-topic of the domain. In many cases, already existing material can be imported as support knowledge in different shapes (e.g., texts, tables, images). Often, the structure of existing documents to some extent can be retained if it is comprehensible and familiar to the domain specialists.

2. **Arrangement of formal knowledge:** The support knowledge covers the scope of the necessary domain knowledge in an informal way, only being meaningful for humans. To actually form an computer interpretable knowledge base, the formal knowledge (using the markup language) needs to be inserted. While complying to the given markup language there is still a large degree of freedom how to order and partition the statements within the documents. For comprehensibility it is reasonable to interweave these parts topically with the support knowledge. Then the support knowledge serves as justification and documentation of the formal knowledge.

3. **Syntactical structure of the formal knowledge:** The users have to use the markup language supported by the system to effectively create the components of the knowledge base. However, considering the overall project scope, the tool developer can extend the tool for supporting different knowledge markups, if beneficial for the use by the knowledge base authors. Knowledge markups can be designed in many different ways strongly affecting readability, writability, or granularity, possibly incorporating peculiarities of the current project or domain.

Freedom in this context means, that in each of these three aspects the document base can be changed without affecting the compiled version of the knowledge base, as it will contain the same entities. In traditional GUI-based knowledge acquisition environments, the adjustment to the conceptual model is achieved by a custom design of the graphical user interface. A document-centered knowledge authoring environment can be customized towards a conceptual model by adjusting the three degrees of freedom discussed above. The three dimensions can be considered to form a kind of state space as shown in Figure 3, describing the possible transformations of a document corpus. A particular point in that

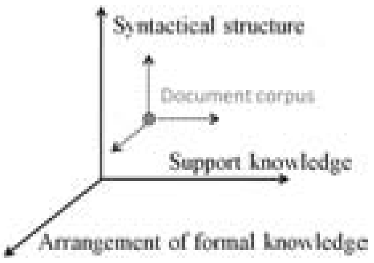


Figure 3: The document space allowing for customization along the three degrees of freedom.

space we call a *document-centered Knowledge Acquisition Architecture* (KAA), determining the scope and arrangement of the support knowledge (1), the arrangement of the formal knowledge (2) and its syntactical structure (3). A fictive document-based KAA, that is optimally accessible by the involved domain specialists, considering readability, comprehensibility, writability and navigability, is contained in this space, but not necessarily easy to find. Hence, the identification of the optimal point is the challenge of customized knowledge acquisition using the document-centered approach.

After all, the document-centered authoring approach also bears some specific challenges when compared to the GUI-based one, that need to be considered. Those are efficient authoring assistance, navigation and search, refactoring, and redundancy detection.

3 The Meta-Engineering Process

The identification of an optimal (or at least appropriate) KAA is non-trivial. Therefore, in this section we introduce a strategy to accomplish this task. We present the meta-engineering process that helps to explore, specify, implement, and use an appropriate KAA for a given project. The evolution of the KAA strives to optimize the criteria understandability, maintainability, and acquisition efficiency. We presume, that a general (extensible) document-centered knowledge authoring tool is initially already existing, which provides some means of knowledge formalization (markups, testing capabilities). The evolutionary process affects both, the document base (content level) and the authoring environment itself (system level) in parallel. Figure 4 shows the meta-engineering process, comprising the key activities exploration, design, and implementation. After the process has been initialized by the exploration phase, alternating design and implementation activities are carried out. The actual knowledge acquisition process runs in parallel. The process is driven by iterated cooperative sessions involving the knowledge engineers and the domain specialists. In the following, we describe the phases in more detail:

Exploration phase At the very beginning a small (but representative) subset of the domain knowledge is selected. For this part of the domain the knowledge engineers create

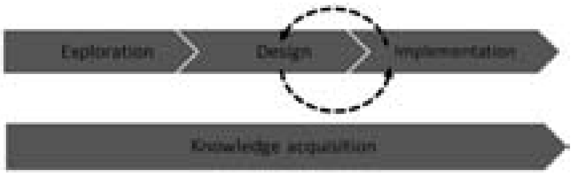


Figure 4: The phases of the document-centered meta-engineering process.

small prototypes according to one (or multiple) ad-hoc defined KAAs using markups already provided by the tool. The goal of this phase is making the domain specialists familiar with the idea that the subject domain knowledge will be managed as documents with knowledge markups, that can be modified using the authoring environment. This phase gives the domain specialists a feeling for document-centered knowledge acquisition, even though the resulting knowledge base might not be very effective and the initially existing markups may be not optimal for this task. For the demonstration effect an actually working toy knowledge base should be created (and played with).

Design phase In the design phase a KAA, tailored to the projects needs, is proposed based on the experiences made in the exploration phase. This first candidate is the starting point of the evolutionary refinement driven by the agile process. Changes and extensions are discussed by domain specialists and knowledge engineers in cooperation. The first two aspects of the KAA (support knowledge and arrangement of formal knowledge) can be evolved intuitively by modifying the existing content (e.g., using simple cut&paste or script-aided) to find a comprehensive organization of the knowledge. However, the specification of the third aspect (syntactical shape) is playing a key role in document-centered meta-engineering: The definition of a suitable knowledge markup language for the knowledge to be captured. Such custom knowledge markup needs to satisfy the following requirements: (1) Allow for the unambiguous translation of the captured knowledge base entity to the executable knowledge repository; (2) Allow for intuitive and simple authoring and comprehending; (3) Allow for simple and seamless embedding into the informal content of the documents; The (creative) task of finding the optimal markup is highly project dependent and has to be performed in close cooperation with the domain specialists. However, it is reasonable to look for domain specific notations, in the seed of the document corpus and the general subject domain, that can be adapted to a formalizable markup. Further, general design principles are known for the design of *domain specific languages* (DSLs) in software engineering, for example proposed by Spinellis et al. [Spi01] and Karsai et al. [KKP$^+$09]. According to Fowler [Fow10] the expressiveness of a DSL is a notable source of errors. Designing DSLs with minimal expressiveness, particularly tailored for a certain purpose, will reduce error rates and therefore improve overall productivity. Fowler also emphasizes that well designed DSLs improve communication with domain experts.

After a markup has been designed and verified according to those general design principles, its applicability should be tested using it on a set of documents of the current content. While a newly designed markup is still unrecognized by the system, it nevertheless can be used in the documents to get an impression of its handling, its readability, and its integration with the content. As this specification and assessment process does not imply any implementation efforts on system level (yet), multiple candidates of different markups can be 'tested' this way easily. If a markup has been assessed as appropriate by the involved contributors, it can be included into the specification of the project's KAA.

Implementation phase When a promising candidate for a suitable KAA (or a small new fragment of it) is specified in the design phase, new features need to be introduced, typically including parsers and editing assistance for the new markups. However, at any time the basic functionality of the authoring environment (read, browse, and edit documents) is provided, independently of the state of implementation of new features. That way, knowledge acquisition can be continued while the implementation of the markup processing functionality is not yet completed. A feature by feature approach, as proposed by many agile software engineering methodologies (e.g., [BA04]), including frequent system updates is appropriate for this customization effort. New completed features will be available after system updates and new markup is automatically applied to the content of the documents. As a reasonable strategy, at first editing assistance, in particular syntax check, for new markups should be addressed to facilitate contributions using it. Then, compilation to the executable knowledge base and code-completion can be added.

Knowledge Acquisition The process of knowledge acquisition itself is not strongly determined by the meta-engineering process. The evolved KAA only describes the content format of the knowledge but does not specify any details of the acquisition process as it is suggested by various knowledge acquisition process models that can be found in the literature. Many of these can be applied in combination with the meta-engineering approach with only minor adaptation.

The document-centered meta-engineering process allows for knowledge acquisition right on from project beginning, however with a not yet entirely customized tool. At any time, content can be contributed and should be inserted according to the most recent version of the project's KAA. With the progress of the design and implementation activities a thoroughly tailored knowledge engineering environment is evolved, empowering the domain experts to contribute the relevant knowledge autonomously to a large extend.

The change of the project's KAA at a medium stage of the project sometimes requires transformation of the content already existing in the document base at that point. To support this task efficiently, efficient script-based methods are required to perform that transformation of the content with respect to structure and syntactical shape.

4 Case Study: Ancient Greek History with HermesWiki

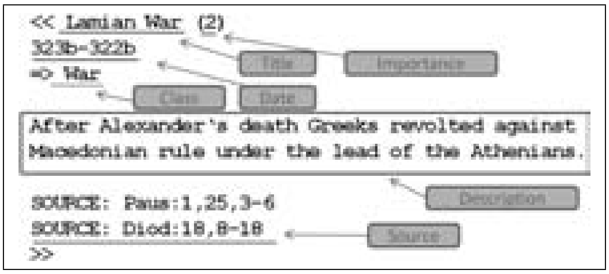
The HermesWiki [RLB+10] is an e-learning platform in the domain of Ancient Greek History implemented as a semantic wiki. It is developed in cooperation with the Department of Ancient History of the University of Würzburg, Germany. The content in general comprises four different types of entities: (1) Medium sized essays, each describing an important topic of the domain as plain text. (2) Important events with date information, a brief plain text description and (historical) source references. (3) Descriptions of important domain concepts (e.g,. persons, cities, islands, habits). (4) Historical sources (German translations). The goal of the knowledge engineering process was the enhancement from a standard wiki (allowing for reading, browsing and plain text search) to a semantic wiki platform providing augmented visualization of the content, interactive features, and semantic navigation and search methods based on a formalized model of the knowledge.

At first, we introduce the current KAA of the HermesWiki platform before the different phases of the meta-engineering process within this project are discussed:

1. Support knowledge: The HermesWiki gives an overview of all the important concepts of the domain, such as persons, cities and peoples. Each so called glossary concept is briefly described on a distinct page. However, the most important content parts are the essays, each covering some important aspect of ancient history by a coherent description.

2. Arrangement of formal relations: The HermesWiki ontology is entirely defined in the wiki. General terms and relations of classes and properties, e.g., the class hierarchies, are defined on a few centralized pages containing the vocabulary definitions. Instances and their interrelations however, are widely distributed over the wiki, being strongly interwoven with the support knowledge according to the domain context. Considering the glossary concepts, the general attributes, such as birth and death dates of persons or coordinates for locations, are defined on the corresponding wiki page. The time events, forming the most important entities of the formal knowledge base, technically can be defined on distinct pages or inline anywhere within the source text of some page (e.g., essay). While it is reasonable to have own pages for very important events, we also perceived the inline definition for events of medium/minor importance in the context as practical (as it allows for very quick definitions). However, an event defined inline later can be easily extracted to a new page by a refactoring operation.

3. Syntactical Structure: The syntactical shape of the class hierarchy discussed above is a so called *dash tree*, that proved to be practical for concise representation and quick editing abilities. Any term being a dash-tree-child, i.e., follows with an incremented number of dashes, is defined as a subclass of its parent. Another important (customized) formalization aspect of the HermesWiki KAA is the markup for the inline definition of time events. Figure 5 shows a markup example for the time event *Lamian War*. The markup is translated and added to the designed ontology. As the first information the title of the event (*Lamian War*) is given, followed by the importance rating defining its relevance for student exams. In the next line, the timestamp of the event is notated, also including annotations for different degrees of uncertainty. Then, introduced by “=>” an optional class membership definition can be added. Further, the body of the markup follows, consisting of a (freetext) description of the actual event. The markup concludes with an (optional) list of historical sources where the event is mentioned, explicitly marked by the keyword



```
<< Lamian War (2)
323b-322b
=> War
After Alexander's death Greeks revolted against
Macedonian rule under the lead of the Athenians.
SOURCE: Paus:1,25,3-6
SOURCE: Diod:18,8-18
>>
```

Figure 5: The markup to formalize time events 'inline' in HermesWiki

"SOURCE:" as the first word of a new line (from *Diodor* and *Pausanias* in this example).

In the following, we sketch the progress of the phases of the meta-engineering process:

- **Experimental phase:** At the beginning of the project different ways of structuring the content entities (essays, events, concept-descriptions, sources) were discussed. It became obvious, that the domain concepts (e.g., cities) should be described independently of the essays referring on them. Describing the concepts in a very general way on distinct articles allows referencing from different contexts/essays. As opposed to this, specific events should not necessarily require an own page, but flexible definition 'inline' within an article (i.e., essay) proved to be appropriate.

- **Design phase:** In the design phase at first a simple markup for defining time events, similar to the one shown in Figure 5, was specified in a workshop together with the historians. The ability to specify a class membership was not contained in the first version of the markup used for months. As these memberships then showed to be required and had to be inserted using the general rdf-turtle-syntax[3] additionally, the time event markup was extended accordingly. For efficient navigation and search a taxonomy of domain concepts was necessary. To allow for simple and quick modification of the class hierarchy for the domain experts, we decided for a dash-tree markup for defining classes and subclass relations.

- **Implementation phase:** In general, the web application is frequently updated when features are added or improved. The time event markup was implemented in multiple stages. First, the events only have been recognized and highlighted in the page view. In the next step its translation to the RDF-store was carried out to make the knowledge available for automated processing. Recently, the extension for defining the class membership has been introduced.

- **Knowledge acquisition:** One advantage of the meta-engineering process is the possibility to start knowledge acquisition at an intermediate stage when the development of the tool is not yet finished. Large parts of the content (i.e., essays, concept descriptions) could be evolved independently of the current implementation state. While the design of the KAA is developed in close cooperation of knowledge engineers and domain experts, large parts of the knowledge acquisition and formalization

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

in this project is in general performed by the domain experts autonomously. However, axioms and entities describing the terminology of the ontology are developed in close cooperation.

More details about the HermesWiki ontology and its use cases can be found in [RLB$^+$10].

5 Prototype Implementation

In this section, we introduce the document-centered authoring tool *KnowWE* [BRP11] and a corresponding metatool that allows for the implementation new custom-tailored knowledge markups for KnowWE at low engineering costs. KnowWE is based on a standard wiki engine, providing web-based access and user management for collaborative authoring. Every wiki page corresponds to one document of the document-centered approach and is edited using the well-known and established wiki interface. After each modification of a wiki page, it is processed and segments complying to the knowledge markup are compiled into the knowledge base (if they do not contain errors). Currently, there are markups and repository connectors implemented for *d3web*[4] and RDFS/OWL. However, this is no restriction, as the approach is independant of the employed formal knowledge representation, as new knowledge base repositories can be integrated easily. Figure 2 shows an example wiki page using some markup for rules in d3web.

The KnowWE Metatool The aim of metatools for knowledge acquisition is to reduce (software) engineering costs for building customized knowledge acquisition tools. In the context of document-centered knowledge acquisition, the engineering task requires to implement the custom knowledge markups (DSLs) to be translated to the knowledge base repository. To allow for efficient and autonomous editing, user assistance, such as syntax checks, error highlighting, or code-completion, needs to be incorporated. The KnowWE metatool allows to define new markup by creating a graph-based schema structure using regular expressions, cardinality constraints, and predefined entity types. Still, the task of creating a markup language using the KnowWE metatool is non-trivial. However, it is carried out by a system developer without involving domain specialists or domain knowledge, once a specification of the required markup is given. Being used to the metatool and the way the schema definition is processed by the KnowWE architecture, developers can implement new markups very quickly. Figure 6 shows the implementation of the HermesWiki time event markup [RLB$^+$10] within the KnowWE metatool. The expressiveness with respect to language complexity in principle is restricted using this mechanism. However, simplicity is a major design objective of the created languages in this context and according to our experiences up to now, all intended markup languages could be modeled using this technique.

The metatool, while still being in an early stage, compiles these kinds of schema definitions to a KnowWE plugin containing a full-fledged implementation of the markup, which

[4]http://www.d3web.de

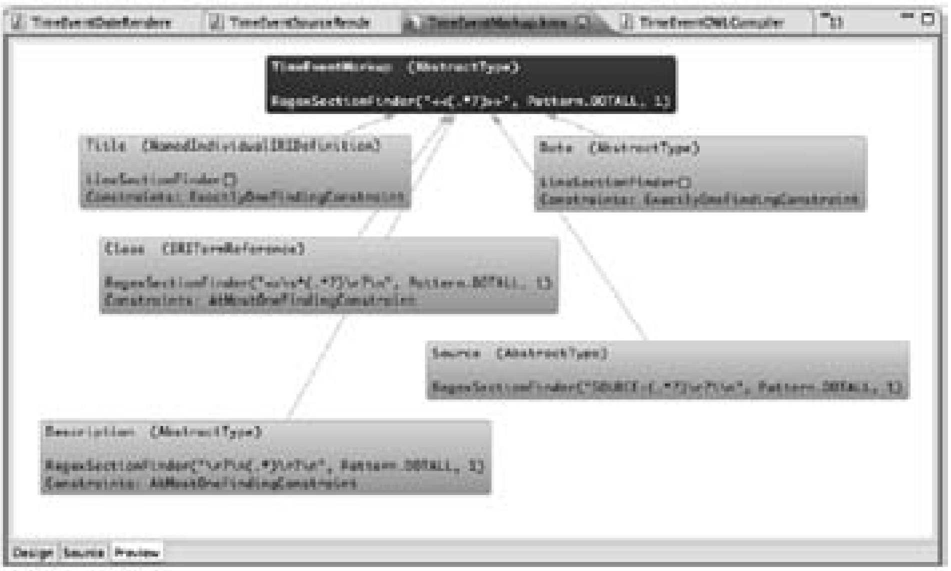
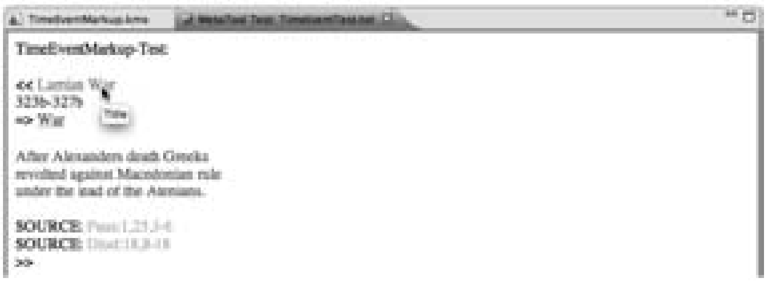

TimeEventDateRenderer
TimeEventSourceRende
TimeEventMarkup.kdom
TimeEventOWLCompiler
TimeEventMarkup (AbstractType)
RegexSectionFinder("<<(.*?)>>", Pattern.DOTALL, 1)
Title (NamedIndividualIRIDefinition)
LineSectionFinder()
Constraints: ExactlyOneFindingConstraint
Date (AbstractType)
LineSectionFinder()
Constraints: ExactlyOneFindingConstraint
Class (IRITermReference)
Constraints: AtMostOneFindingConstraint
Source (AbstractType)
Description (AbstractType)
RegexSectionFinder("\r?\n(.*)\r?\n", Pattern.DOTALL, 1)
Constraints: AtMostOneFindingConstraint
Design
Source
Preview

Figure 6: The time event markup of the HermesWiki project defined in the KnowWE metatool.

only needs to be copied to the plugin-folder of a KnowWE installation. It can then be used on any document/page in any number or order to populate the knowledge base. This includes parsing, reference resolution with errors handling and correction propositions (based on edit-distances to existing terms). In particular, the closed-world knowledge compilation problem is solved incrementally, guaranteeing high performance compilation even for very large knowledge/document bases [RSLP11]. A detailed explanation of the KnowWE architecture and the functionality of the schema processing is beyond the scope of this paper. However, apart from a few lines of script, actually inserting the knowledge into the knowledge base repository (RDF-triples in this example), the plugin is entirely defined by that schema. The metatool is implemented as a plugin of the eclipse framework[5]. To allow for quick feedback cycles within the markup development process, we included a quick test mechanism. The developer can specify a test document/wiki page containing examples of the target markup to be implemented. After triggering the quick test mechanism the markup schema is compiled and an embedded KnowWE engine is launched, processing the specified test page. The result of the processing is visualized using colors and mouse-overs. Figure 7 shows a test page with above markup schema specification launched. The red underline at *War* indicates that the compilation algorithm has not found a proper definition of the term 'War' in the current document base (only consisting of these few lines in this quick-test example).

A categorization for knowledge acquisition metatools is given by Eriksson et al. [EPG+95], distinguishing *method-oriented*, *architecture-oriented*, and *ontology-oriented* metatools.

[5]http://www.eclipse.org



```
TimeEventMarkup-Test

<< Lamian War
323b-322b
=> War

After Alexanders death Greeks
revolted against Macedonian rule
under the lead of the Atenians.

SOURCE: Paus 1.25.3-6
SOURCE: Diod 18.8-18
>>
```

Figure 7: The time event markup in the quick test view.

According to this, the presented tool is an architecture oriented metatool. The KnowWE core and the metatool are available as open source LGPL-licensed[6].

6 Related Work

The idea to build customized domain-specific knowledge acquisition tools to enable domain specialists to become directly involved in the knowledge encoding process more easily has been studied before [MCW+86, Gal87]. The resulting tasks of specification and implementation of such tools have been addressed by establishing a conceptual model of the domain in advance followed by the actual tool design and implementation accordingly [Mus88]. Meta-level tools support the knowledge engineers on these implementation tasks [Gap91, Eri92, EM93]. The meta-level tool Protégé[7] was designed to allow knowledge engineers to specify a model of the domain. Based on that model a knowledge acquisition tool is generated automatically [Mus89]. Protégé has been evolved consequently for years [GMF+03] incorporating problem-solving methods and application tasks.

All this work was focused on graphical user interfaces. Document-centered authoring using knowledge markups is a fundamentally different approach for authoring formal knowledge bases. However, the basic idea, that customization according a conceptual model shared by the domain specialists significantly improves their interaction capabilities with the knowledge base, is similar. A meta-level approach for domain or project specific customization of this document-centered approach has not yet been addressed. Also the technical challenges for a meta-tool in this case significantly differ from those mentioned above. Parsing and compiling knowledge markups into a knowledge base repository with user assistance is more related to the discipline of compiler construction in software engineering [ALSU06]. For the parsing task a class of tools called *parser generators* have been developed (e.g., ANTLR[8]), generating parsers from a declarative specification (grammar) of the language. Such a tool can be considered as partial metatool for analyzing grammar-

[6]https://isci.informatik.uni-wuerzburg.de
[7]http://protege.standford.edu
[8]http://www.antlr.org/

based formal languages, as parsing is only the first step of the overall compilation task. However, the problem of compiling knowledge has turned out to be significantly easier than compiling general purpose programming languages and partly can be solved by a general mechanism (assuming the knowledge base to have set characteristic, c.f. [RSLP11]).

In the context of graphical user interfaces the entire specification and implementation of the tools needs to be completed before actual knowledge acquisition activities can be started by the domain specialists. In the document-centered context knowledge acquisition can be started right at the beginning by editing documents using the basic document authoring environment. Another advantage is, that the final design details of the specification of the optimally customized knowledge acquisition environment can be delayed to a later point in the project. The specification of a suitable tool complying to a conceptual model is a challenging task in the document-centered as well as in the GUI-based approach. However, the document-centered approach allows to drive an agile process, that optimizes this specification during the project progress, ensures larger flexibility and helps to prevent wrong design decisions at very early stages.

7 Conclusion

In this paper, we introduced a knowledge engineering approach that aims at the development of custom domain-specific knowledge acquisition tools for document-centered development. Opposed to knowledge acquisition with graphical user interfaces, in the document-centered approach the domain specialists edit documents using knowledge markup languages to form the actual knowledge base. We presented the meta-engineering process that helps to specify and implement a suitable customized authoring environment that optimally complies with the conceptual model of the domain specialists. The agile process allows for knowledge acquisition and customization process to run in parallel and coordinates the introduction of new domain-specific knowledge markups. We presented the document-centered knowledge acquisition environment KnowWE and a corresponding metatool that allows for quick and simple implementation of new (domain specific) markups, specified within the meta-engineering process. The practical use of the overall approach was illustrated by a real-world case study within the HermesWiki project. Currently, we are applying the approach also in other projects to gain more experiences in the design of helpful markup languages for different domains and project scopes.

References

[ALSU06] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2006.

[BA04] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley, Boston, 2004.

[BRP11] Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe. KnowWE: A Semantic Wiki for Knowledge Engineering. *Applied Intelligence*, 35(3):323–344, 2011.

[EM93] Henrik Eriksson and Mark Musen. Metatools for Knowledge Acquisition. *IEEE Softw.*, 10:23–29, May 1993.

[EPG$^+$95] Henrik Eriksson, Angel R. Puerta, John H. Gennari, Thomas E. Rothenuh, Samson W. Tu, and Mark A. Musen. Custom-Tailored Development Tools for Knowledge-Based Systems. Technical report, Stanford University School of Medicine, 1995.

[Eri92] Henrik Eriksson. Metatool support for custom-tailored, domain-oriented knowledge acquisition. *Knowledge Acquisition*, 4(4):445 – 476, 1992.

[Fow10] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[Gal87] William A. Gale. Knowledge-based knowledge acquisition for a statistical consulting system. *International Journal of Man-Machine Studies*, 26(1):55 – 64, 1987.

[Gap91] Ute Gappa. A tool-box for generating graphical knowledge acquisition environments. In *Proceedings of the World Congress on Expert Systems*, pages 797–810, 1991.

[GMF$^+$03] John Gennari, M. Musen, Ray Fergerson, W. Grosso, Monica Crubezy, H. Eriksson, Natalya Noy, and Samson Tu. The evolution of Protege: an environment for knowledge based systems development. *Int. J. Hum.-Comput. Stud.*, 58(1):89–123, 2003.

[KKP$^+$09] Gabor Karsai, Holger Krahn, Class Pinkernell, Bernhard Rumpe, Martin Schneider, and Steven Völkel. Design Guidelines for Domain Specific Languages. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling*, pages 7–13, 2009.

[Mar09] R. Martin. *Clean Code: Handbook of agile software craftsmanship*. Prent. Hall, 2009.

[MCW$^+$86] Mark A. Musen, David M. Combs, Joan D. Walton, Edward H. Shortliffe, and Lawrence M. Fagan. OPAL: Toward the Computer-Aided Design of Oncology Advice Systems. In *Computer Application in Medical Care*, pages 43–52, 1986.

[Mus88] M. A. Musen. Conceptual Models of Interactive Knowledge-Acquisition Tools. In J. Boose, B. Gaines, and M. Linster, editors, *Proc. of the European Knowledge Acquisition Workshop (EKAW'88)*, pages 26–1 – 26–15. Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, Germany, 1988.

[Mus89] M. A. Musen. An editor for the conceptual models of interactive knowledge-acquisition tools. *Int. J. Man-Mach. Stud.*, 31:673–698, December 1989.

[PG92] Frank Puppe and Ute Gappa. Towards knowledge acquisition by experts. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, volume 604 of *LNCS*, pages 546–555. Springer, 1992.

[RLB$^+$10] Jochen Reutelshoefer, Florian Lemmerich, Joachim Baumeister, Jorit Wintjes, and Lorenz Haas. Taking OWL to Athens – Semantic Web Technology takes Ancient Greek History to Students. In *Proc. of the 7th Extended Semantic Web Conf.*, pages 333–347. Springer, 2010.

[RSLP11] Jochen Reutelshoefer, Albrecht Striffler, Florian Lemmerich, and Frank Puppe. Incremental Compilation of Knowledge Documents for Markup-based Closed-World Authoring. In *K-CAP '11: Proceedings of the sixth international conference on Knowledge Capture*, pages 81–88. ACM, 2011.

[Spi01] Diomidis Spinellis. Notable Design Patterns for Domain Specific Languages. *Journal of Systems and Software*, 56(1):91–99, February 2001.