Loki – the Semantic Wiki for Collaborative Knowledge Engineering

Krzysztof Kutt* and Grzegorz J. Nalepa

Jagiellonian Human-Centered Artificial Intelligence Laboratory (JAHCAI) and Institute of Applied Computer Science, Jagiellonian University, 31-007 Kraków, Poland
E-mails: krzysztof.kutt@uj.edu.pl, gjn@gjn.re

Abstract. We present Loki, a semantic wiki designed to support the collaborative knowledge engineering process with the use of software engineering methods. Designed as a set of DokuWiki plug-ins, it provides a variety of knowledge representation methods, including semantic annotations, Prolog clauses, and business processes and rules oriented on specific tasks. Knowledge stored in Loki can be retrieved via SPARQL queries, in-line Semantic MediaWiki-like queries, or Prolog goals. Loki includes a number of useful features for a group of experts and knowledge engineers developing the wiki, such as knowledge visualization, ontology storage or code hint and completion mechanism. Reasoning unit tests for knowledge quality assertion are also introduced. The paper is complemented by the formulation of the collaborative knowledge engineering process and the description of experiments performed during Loki development to evaluate its functionality. Loki is available as free software at https://loki.re.

Keywords: Knowledge engineering, Semantic wiki, Software engineering, Unit tests, Prolog

1. Introduction and Motivation

In recent years, a shift from the traditional knowledge engineering (KE) to the group-based KE has been visible. This shift corresponds to the transformation from process in which the main role was played by knowledge engineers, to a group effort, in which domain experts are main contributors [1]. This is in line with the emerging movement of knowledge socialism, which promotes the development of technologies that facilitate the exchange of knowledge between different stakeholders for the public good [2]. The research literature provides several terms for such group-based KE process: collaborative KE, distributed KE, collective KE, and cooperative KE. There are no well-established definitions of what they mean [3], and experts can often understand them in different ways [4], or use some of them interchangeably (e.g., distributed KE and collaborative KE in [5], or collaborative KE and collective KE in [6]). However, they should be distinguished based on the characteristic features indicated by experts and supported by dictionary definitions. Making this distinction transparent, allows to differentiate various situations where multiple users are involved in the creation of knowledge. It will also support us in the specification of the context of this work:

- Distributed KE is a general term that describes KE process partitioned into a number of users working together on one knowledge base by using different terminals [7, 8]. This definition is similar to the notion of “distributed systems”, which covers a wide range of different systems in which communication is provided by sending messages over the network [9].

- Cooperative KE is a process, in which the problem is divided into separate fragments, like in “divide and conquer” approach. Each participant is responsible for one part [10]. They carry out their own goals, but are willing to help others [11]. They ultimately formulate knowledge, which brings benefits to each of them [12]. An example of such a process can be found in HermesWiki [13], where tasks are divided between students who can work on them independently.

*Corresponding author. E-mail: krzysztof.kutt@uj.edu.pl.
Collaborative KE is a joint involvement of participants in the project for a common purpose, although it may result from different motivations. In this scenario, users operate on the same piece of the problem, associated with the occurrence of conflicts and dynamics of opinions [14, 15]. Collaborative KE examples are ACKTUS-dementia [16] and Dispedia [17] where patients and doctors share knowledge about the same diseases, as well as Turkic thesaurus [18] developed by group of engineers and domain experts.

Collective KE is done by a whole group of people, who aim towards a common goal, have similar motivations and socio-economic interests. Members of the group are equal, which means the similar level of skills [11, 19]. Examples of this group are Caucasian Spiders database [20], International Classification of Diseases (ICD-11) [21] and OpenResearch.org semantic wiki [22]. The projects are developed by a group of similar participants (biologists, doctors and researchers) with a shared goal, using their own resources.

Our work is set within the collaborative knowledge engineering (further referred to as CKE). It is the most common variant of the group-based KE. However, it poses many challenges that still await for a proper solution and require appropriate supporting tools [4]. It is also worth noting that collaborative KE and collective KE differ primarily on levels of organization and communication, so probably collective KE problems may be similar to the ones in collaborative KE and proposed solutions will also apply to collective KE.

CKE usually requires the creation of diverse groupware tools supporting it [23, 24]. In our work we argue that one of the most suitable tools for CKE are semantic wikis. They are built on classic wiki systems that have gained immense popularity, because of their usability, brought by easy installation and maintenance without a long introduction training [25]. Wikis are pretty simple and robust systems for collaboration, as indicated by the relatively high quality of the Wikipedia project [26]. They were further extended by the use of the Semantic Web technology. The term "semantic" is here understood as a meaning of information [27], and as a machine-processability of the information [28], which will also enable information exchange between the agents [27]. Incorporation of selected technologies, like URIs for entity identification [27, 28], RDF for knowledge representation [29], OWL for more sophisticated relations in a form of an ontology [30], and SPARQL for querying the knowledge base [31] extends the capabilities of the wiki systems for the automatic processing of knowledge contained in the wiki and for conducting inference. Such enriched wikis, called semantic wikis, became useful tools for CKE: on the one hand, easy to use, and on the other, offering the ability to process knowledge at a certain level of formalization [32].

Furthermore, we observe that today CKE can harvest on the decades of software engineering (SE) research [33]. We believe this is due to the fact, that number of the SE methods at tackling similar challenges to those of CKE. This includes the gradual and dynamic group-based development process. This is why in our work, we propose a specific CKE process that organizes the main KE phases and extends them with number of SE methods. Furthermore, on the level of the groupware tool supporting the CKE process we use selected techniques established in the practical SE.

In the above mentioned context, the objective of this paper is twofold. First of all, we introduce a specific and detailed workflow or process for CKE. In this specification we combine the methods of classic KE with the above mentioned techniques of SE. Second of all, we present the Loki groupware tool, which is a semantic wiki we developed for supporting our CKE workflow. It is targeted at a group that consists of knowledge engineer(s) and domain experts with short training in specific knowledge representation techniques used in a project. The choice of specific solutions for this system was based on our assumption that specific solutions developed recently in SE can be useful for addressing current KE issues. Among such practical solutions there are, e.g., agile methodology and unit tests. Finally, to make our considerations more specific, in this paper the KE process is narrowed to a group that organizing our CKE workflow. It is targeted at a group that consists of knowledge engineer(s) and domain experts.

The rest of the paper is organized as follows. In Sect. 2 we discuss the requirements for tools that support collaborative knowledge engineering. Then, we provide the specification of our CKE process in Sect. 3. The main part of the paper, i.e. the presentation of the Loki wiki, is placed in Sect. 4. We discuss the use of reasoning tests for controlling the quality of the knowledge base in Sect. 5. The experiments we performed during Loki development are described in Sect. 6. Related works and comparison with existing wiki systems in the context of CKE are discussed in Sect. 7. Summary and future works are given in Sect. 8.
2. Requirements for CKE

Experts in knowledge engineering seem to agree that specific use cases of collaboration require adequate tools. Therefore, it is not possible and there never will be unanimously accepted universal tool for all possible situations. On the one hand, there may be cases that require a specific change acceptance protocol and on the other hand the situations in which everyone can make immediate changes [4]. There is, however, possibility to specify the list of common requirements that should be addressed in order to provide proper CKE tools. There were many attempts to define such a list [4, 15, 34–41]. Each of them paid attention to other aspects of the problem. We have reviewed them and eventually partitioned them into six fields of CKE support [23]:

– Agile CKE process – there is a need for mature agile methodology for CKE, as it still has not been developed [42], in particular:

R1. Development should be fast and done in a robust agile way,
R2. Different roles of users involved in the process should be identified and supported in a proper way,
R3. There should be a possibility for users to specify expectations which will have an impact on the direction of the project’s development.

– Supporting tools – CKE systems have to be built on the base that allows for collaboration:

R4. Provide compatibility with mainstream tools, standards and methods, especially domain specific ones,
R5. Manage current KB status and make it possible to download the stable version of KB,
R6. Adapt the system to specific project requirements.

– Knowledge representation and reasoning – as domain knowledge should be managed by domain experts, not knowledge engineers, there is a need to develop a suitable knowledge representation:

R7. The representation should be adjusted specifically to the project and group that will use the CKE tool and should be both easy to use for experts and powerful in terms of inference capabilities.

– Quality management – there is a need for assuring the proper level of knowledge base quality:

R8. Provide automatic methods for knowledge verification, e.g. knowledge consistency check or execution of automatic tests,
R9. Determine the credibility of users expertise and sources of knowledge, especially in open environments where everyone can make changes,
R10. Keep in mind that users are domain experts: give them a possibility to approve, disagree or discuss with others, but also be prepared for resolving user conflicts.

– Change management – management of factual changes in the collaborative setting is more difficult than in the classical KE:

R11. There is a need for robust versioning mechanism that takes care of different types of changes and covers various characteristics of the process.

– User involvement – as participation in CKE process is not spontaneous, various incentives should be considered to motivate people:

R12. Take care about usability, i.e. powerful and easy to use interface, compatibility with established practices, proper visualisation and automation capabilities.
R13. Consider usage of gamification techniques. However, one should keep in mind that some users prefer “hard work”, and they do not find game elements motivating at all.

This list has set the scope of the work undertaken to adjust an existing semantic wiki system to the needs of the collaborative process of knowledge engineering. However, in order to achieve this goal, next we provide a complete specification of a CKE process.

3. Formulation of the CKE Complete Process

3.1. CKE Process Description

The proposed process of collaborative knowledge engineering is the result of analyzes of existing knowledge engineering methodologies [43–47]. Intermediate results of these analyzes were presented earlier [48]. They were then combined with the inspirations from the agile methodology in SE, in particular Scrum [49, 50]. An important role was also played by the general outline of the CKE agile process proposed in [51], which is refined and clarified here. Finally, the
CKE process consists of 9 steps grouped into 4 stages (see Fig. 1) performed in multiple iterations until the expected state of the system is reached:

1. **System metaphor**: each iteration begins with the *motivating scenarios* step to regularly discuss the use cases and scenarios and to match them to the current needs. The *competency questions* are also specified here. They can be written down formally, e.g. in a form of reasoning unit tests.
2. **Planning game**: the *iteration planning* step is explicitly introduced into the proposed methodology to reflect an important step in the agile process. During this step, specific tasks are created, estimated and selected for the current iteration.
3. **Implementation**: the most important stage of the iteration is actual integration of knowledge into the system. It covers the whole process that starts with *knowledge acquisition* from experts, domain texts and other resources. The gathered knowledge is then *conceptualized* and the *integration* with the existing well-established vocabulary is considered. Finally, the knowledge is *implemented* using selected formalized system, e.g. in a semantic wiki.
4. **Integration**: each iteration is concluded by a proper *evaluation* of the knowledge base. In addition to the verification of knowledge at the formal level, it should also be checked at the conceptual level to check how reliably it represents the domain, e.g. using the reasoning unit tests specified previously. The entire process is supplemented by documentation in a form of e.g. competency questions and reasoning unit tests.

During the initial iterations of presented process, a common language is established [47, 52], and later, in the next ones, knowledge is created from less to more formal. This is achieved by a team in which specific roles can be distinguished: *the product owner*, who represents the client and has the whole system vision, *the CKE process master* (named as scrum master in Scrum methodology), who oversees the project course, is responsible for the proper implementation of the process, ensures that all obstacles preventing the team from completing the project are removed, *the team of domain experts* and 1-2 *knowledge engineers*, who support the team. One can also distinguish some roles within the team: *the adders*, who create a lot of material, but not always well semantically marked, *the synthesizers*, who take care about semantics and concepts interrelations, *the cops*, who are responsible for imposing standards and schemes and then they are monitoring them (cf. [53]).

### 3.2. European Wiki Case Study

Sample *European Wiki* project is introduced here to present the above proposed process in action. It is developed by a group of colleagues that will work online to develop the wiki with description of all European countries. It will be used by them as a supporting system during their travels across the continent. As there is no “client” in this setting, one person from the group has been selected as a product owner, who will watch over the main purpose of the project. Also, one of them was selected as a CKE process master that will take care of the whole process. Finally, all of them except the product owner will work as a development team that will build the wiki. To simplify the description, only the first iteration will be presented here, along with the KB made of three wiki pages prepared by two users (*kkutt* and *yoda*).

1. The process begins with the definition of *motivating scenarios*. During this step, three use cases for
the European Wiki were proposed: (a) I want to see cities: A, B and C. In what order do I have to visit them to be able to use direct flights? (b) I want to see all European capitals. List all of them. (c) I am in city A. What is interesting here?

2. In the next step more specific competency questions were proposed. They were formally specified using the reasoning unit tests.

3. When such system specification is described, the iteration planning step takes place. In the first iteration of the European Wiki project, three tasks were defined: (a) Describe London, (b) Describe Paris, (c) Describe Cracow.

4. After planning game, the actual implementation stage begins. During this block, two users made six changes to the European Wiki project and discussed the difficult points. In the background, the semantic changelog was created and parsed to provide a gamification-based incentives for users. All task-related decisions were marked on an iteration board (a scrum board; a place with a list of all tasks selected for current iteration grouped into three main categories: “to do”, “in progress” and “done”), i.e. user kkutt assigned task (a) to himself and moved it to the “in progress” section.

5. Each iteration is concluded by an evaluation. During this step, reasoning unit tests results are consulted to check the current knowledge quality and the level of requirements fulfillment.

6. Everything is supplemented by a documentation step. Within the first iteration of European Wiki project, the most important action was to comment the first design decision made: cities will be described by a city category. A note appeared in the proper place in the system.

In order for such a process to be successful, it is necessary to develop the right tool. It should allow such a process to be carried out, to facilitate it and, moreover, it should meet the requirements listed in Sect. 2. We argue that the Loki platform is such a system. It is described in the next section.

4. The Loki Platform

4.1. The Architecture of the Loki Ecosystem

Originally, the preliminary version of the Loki (a short for „Logic in wiki” or „Logic-based wiki”) engine was developed by us in 2009 to show the possibility of combining wiki systems with the representation of knowledge in Prolog. As a base, the DokuWiki engine was selected. Loki was created as a plugin for this wiki system [6, 54]. Later on, we extended it subsequent modules, as a part of the PhD thesis [23], leading to the emergence of the ecosystem presented in Fig. 2. It is now composed of five groups of modules. All of them are developed as plugins for DokuWiki that provide additional functionalities for Loki engine.

– *Loki core.* This group represents the necessary minimum needed for the basic functioning of a semantic wiki. The DokuWiki engine manages the basic functions (page editing, user accounts), while the Loki engine handles the processing of knowledge contained in the pages in the form of semantic annotations and in the form of Prolog’s code. It adds the appropriate notation, allows one to query the knowledge base and allows to manage quality in the form of reasoning unit tests. Among core modules there are also RDF-FLoki for knowledge visualization on given page and LokiOntology for storing the simple ontology within wiki with the use of selected concepts from OWL language [30].

– *Knowledge representation modules.* These modules provide a possibility to use additional knowledge representations than the core ones, i.e. text, images, tables, semantic annotations, Prolog clauses. Currently we provide modules for three such representations: Business Process Model and Notation (BPMN; business processes) [55], Semantics Of Business Vocabulary And Rules (SBVR; business rules) [56] and eXtended Tabular Trees [57] (XTT2; production rules) [5].

– *Semantic change graph.* These modules are related to creation (PROV) and visualization (PROV-Viz) of semantic change graph. This is an RDF-based graph that covers various aspects of CKE process in the wiki: statistics of triples added/removed, metrics of KB’s subsequent states, results of reasoning unit tests, rates from wiki users (by RevisionsRater plugin). As an RDF graph, it forms a (meta-)knowledge base that can be further explored (e.g. using Virtuoso) or used as a data source for other purposes.

---

1For documentation and downloads see: https://loki.re/.

2See: https://www.dokuwiki.org/.


ther processed leading to the more thoughtful analyses of CKE process.

– **User-oriented modules.** This group consists of three modules. First one provides gamification elements as incentives for users in a knowledge engineering process. The second module focuses on the development of a UI that adapts to the needs of identified user groups, e.g. adders that creates a lot of content and synthesizers that are aimed at knowledge consistency. Finally, the ScrumIDE module is an attempt to connect wiki with an IDE supporting agile software development.

– **DokuWiki plugins.** Due to the fact that Loki uses the DokuWiki system as a base, one can extend Loki features with all available DokuWiki plugins. They give a possibility to e.g. create multilingual wiki, compile TeX commands on wiki pages or create image galleries inside wiki.

The subject of this paper is to present the current state of the **Loki core**. The description of the remaining modules is out of scope of this paper.

### 4.2. DokuWiki Engine

The DokuWiki system was selected as the foundation for Loki, as it is a very compact and fast implementation of the Wiki concept. It controls the creation of pages and saving them in text files, purely using the Unix filesystem with no need of a database engine. It provides a simple markup, through which one can create headers, links, lists, format text or insert drawings (see Fig. 3). DokuWiki engine handles the whole processing of wiki pages that can be easily extended through the plugins mechanism modifying syntax processing, engine actions, and page rendering. All Loki-related functionalities presented below are implemented as these plugins.

### 4.3. Knowledge Engineering with Loki Engine

For the implementation of Loki, we selected SWI-Prolog, as it is a mature, popular and well-supported open implementation of Prolog. The basic knowledge representation in semantic wikis are semantic annotations. To provide a compatibility with existing solutions, it was decided to use annotations markup similar to the one available in Semantic MediaWiki (SMW), one of popular semantic wikis (see Sect. 7 for description and comparison with Loki). When the wiki page is saved, it is parsed and the annotations are translated into Prolog code for further processing. There are three types of annotations available in Loki. All of them will

---


7See: http://www.swi-prolog.org/.
be depicted with examples from city:london page from European Wiki (see Fig. 3 for full wiki page).

- **Category** specifies the type of a concept described on given page.
  * **Example:** London page describes a city.
  * **Annotation:** `[[category:city]]`
  * **RDF triple:** `:city:london rdf:type :city`.
  * **Prolog statement:** `wiki_category('city:london ','city')`.

- **Relation** describes a connection between two concepts (wiki pages). It is an equivalent to an object property in OWL.
  * **Example:** London is a capital of England.
  * **Annotation:** `[[capitalOf::country:england]]`
  * **Prolog statement:** `wiki_relation('city:london ','capitalOf','country:england')`.

- **Attribute** connects a concept (wiki page) with a literal value. It is an equivalent to a data property in OWL.
  * **Example:** The name of the city is “London”.
  * **Annotation:** `[[name::London]]`
  * **RDF triple:** `:city:london :name 'London'`.
  * **Prolog statement:** `wiki_attribute('city: london','name','London')`.

Annotations can be supplemented by the Prolog code placed directly in the wiki text with the `<pl>` tags. The result is one homogeneous knowledge base that can be queried in three different ways (see Fig. 4):

- **SPARQL query language** [31]. A subset of SELECT, ASK and DESCRIBE constructs is supported. Such queries may be placed in wiki text (see Fig. 4) or may be asked using the built-in SPARQL Endpoint. It allows users to specify queries to a wiki via a simple web interface without the need to create a separate wiki page with the query. Queries can also be remotely sent via HTTP GET Requests. Results may be returned as HTML, JSON or RDF/XML. This functionality is currently a standard that allows data to be retrieved from various RDF-based systems.

- **SMW-like queries.** Loki adopted the query language used in Semantic MediaWiki [58]. It is easier to understand than SPARQL and Prolog, because it offers a syntax similar to semantic annotations used in the wiki (see Fig. 4), but at cost of simpler querying capabilities.

- **Prolog goals.** One can also specify arbitrary Prolog goals within the wiki. They can be used to retrieve knowledge written in the both forms: annotations and Prolog code, as they form one knowledge base.

Besides storing and querying the knowledge in the wiki, Loki also has a functionality to export each page as an RDF/XML file. After pressing the „Export to RDF“ button, all semantic annotations saved on given page are translated into fully RDF/XML compatible document and available for download. More detailed technical specification of Loki engine is available in [59] and on Loki website.

---

8 See: https://loki.re/wiki/docs:userman#sparql_queries.
9 See: https://loki.re.
4.4. Visualization of Knowledge

Knowledge written in the wiki is not visible to the user at a glance. One simply sees the text and links on the page, but does not know which fragments are formally written in the form of annotations. It is possible only when edition form is opened and the wiki text is visible. To facilitate a quick look at the knowledge available on a given page, a visualization module has been created. It is based on the RDF/XML export provided by the Loki engine (see Sect. 4.3). Thanks to the fact that the export function saves the generated RDF/XML file in the wiki file system, it can be easily used later. For the purpose of visualization, such a file is parsed by the jQuery script, which immediately generates a graph in SVG format using \( \text{<svg>} \) tags supported by HTML5. Sample graph generated for London page (see Fig. 3) is presented on Fig. 5.

Generated graphs use different types of nodes to differentiate between different types of elements: current page is represented as a gray node, other wiki pages, i.e. objects of object properties, are shown as green nodes, while literals, i.e. objects of data properties, are blue nodes. All properties are represented by arcs. This representation is consistent with convention used in RDF W3C Recommendations [60], where RDF resources are represented as ovals and literals as rectangles. All oval nodes are links to proper wiki pages, providing a simple and intuitive way to explore the whole knowledge graph by going through the small visualizations on each page.

4.5. Ontology Storage

A separate module for ontology storage was developed for Loki allow to explicit definition of ontology and its structure. Firstly to provide an easy way to store the general idea of ontology used in one place, and secondly to help users with stating new annotations on wiki pages – with an ontology in one place, it is easy to look at the available relations and classes and copy them to new wiki page.

Given the architecture of Loki, we decided to store ontologies on \texttt{special:ontology:\{name\}} pages,
where \{name\} represents the name of the ontology stored on a specific page: default for base ontology, foaf for Friend-of-a-Friend\(^{11}\), etc. In these pages, ontologies are stored using XML syntax. When connected with XSLT stylesheets\(^2\), it offers great visualization capabilities. Two stylesheets were prepared: one with simple edition form, that gives a possibility to enter a new statement or delete an existing one, and the second for viewing the ontology (see Fig. 6).

### 4.6. Code Hint and Completion Mechanism

Code hint and completion mechanism was selected as the most useful extension for Loki, as a solution for a problem that appears in all conducted experiments (see Sect. 6). Namely, typos or mistakes made by users in the semantic annotations led to the emergence of knowledge bases that were not fully useful and required time-consuming debugging. As annotations in Loki are case-sensitive, even mistake \{category: name\} with \{category:Name\} leads to two very different statements.

To overcome the discussed problems, as a part of lokiontology plugin used also for ontology storage within Loki, we implemented a code hint mechanism that follows the rules using JavaScript\(^1\). The mechanism aims at suggesting matching annotations, classes, as well as the available object and data properties. For example, let us consider the ontology presented on Fig. 6. On the current page, user already put the \{category:city\} statement. As a result, directFlightTo, onRiver and other properties are proposed (see Fig. 7). If user selects the former, all city class instances are proposed as objects (see Fig. 7). If the second is selected, user has to write the string value, so nothing is proposed.

### 5. Controlling Knowledge Base Quality with Reasoning Unit Tests

Collaborative knowledge engineering is not only related to the ways of gathering and integrating knowledge or its subsequent processing. The means to ensure the required level of quality of this knowledge are also very important (see Requirement R8 on page 3). In software engineering there are various types of tests to address this issue. Among them there are unit tests, that are used on daily basis to provide a quick quality check with immediate results that will signal potential problems with the code [61].

Such an idea was formally adopted to KE [62] and implemented in KnowWE semantic wiki [63]. It was also incorporated into Loki with further improvements. First of all, it is important to note that these tests can, and should, be created by system users (i.e. domain experts) to reflect the real system expectations. This allows the unit tests to check not only the „abstract“ knowledge quality, but also how well the system actually performs its task accordingly to user requirements. Secondly, within the unit tests module implemented in KnowWE, all tests are thrown into a one “big bag”, making it difficult to use with greater number of tests. In Loki the hierarchy is proposed, so similar tests can be grouped in the nested structure, with more detailed tests placed deeper in the hierarchy. The advantage of this solution lies in the ability to not execute the tests when they are not needed. This means, for example, that when any of the parent space tests fails, it is assumed that currently checked part of the KB has errors, so nested tests are not executed, because they are very likely to fail. A sample tests structure, with indication of whether the test was passed (green), failed (red) or not executed (yellow), is shown on Fig. 8. White nodes represent levels in the hierarchy.

Tests structure within Loki is implemented as a nested namespaces tree with the unittest namespace as a root. Each test within this structure is a separate page that consists of exactly one SPARQL query with a set of assertions that specify restrictions on expected query result. Sample test structure from Fig. 8 may be represented as follows:

```markdown
unittest:unittestsoverview

```

Tests are executed after each change in the wiki. The results are presented on a corresponding page in a unittestresults namespace, e.g. for a test saved in unittest:eu:largesttest, the results are available in the unittestresults:eu:largesttest page. Test summary is generated as a table with their status in the unittestsoverview page (Fig. 9).

---

\(^{11}\)See: http://www.foaf-project.org/.


\(^{1}\)Available in the Loki repository: https://gitlab.geist.re/pro/loki/.
Figure 6. Ontology editor (on the left) and visualization (on the right)

Figure 7. Code hint mechanism in Loki: list of properties for city (on the left), list of possible objects for directFlightTo (in the center) and errors highlighting (on the right)

Figure 8. Sample test structure for European Wiki project
Figure 9. Summary of tests. It represents the structure from Fig. 8

As it was mentioned, each test page is composed of a SPARQL query and a set of assertions. Any SELECT, ASK or DESCRIBE query valid in Loki is acceptable as a test. A set of possible assertions is available for each of the query types (see Tab. 1-3). All of them are added to the test by putting a single line on the test page, accordingly to the scheme:

```
[[unittest_assert_{type}:
  {parameter}:
  {value}]]
```

where: type is an assertion type (see Tab. 1-3)[14], parameter is a name of query column used for comparison or special value, value is a value used for comparison in the assertion. There is also a place for optional comment to document the assertion. Sample query with two assertions taken from European Wiki project is presented on Listing 1.

---

Listing 1: A sample test from European Wiki project

SPARQL Recommendation [31] define also the CONSTRUCT query form which returns an RDF graph. They were not presented here due to the lack of their implementation in Loki. It is worth noting, however, that from the testing point of view, they work in the same way as SELECT queries. The main difference is that the result is not a table, but a graph. Therefore, for these queries, one could use the same set of assertions as for SELECT queries.

6. Experimental Evaluation of Loki

We evaluated selected concepts of CKE, as well as the corresponding features of Loki during six experiments we conducted to evaluate specific hypotheses during modules’ and CKE process’ development. Each of these experiments was aimed at creating a knowledge base in the wiki by a group of students of computer science. They were participants of two courses organized at AGH-UST: “Semantic Web Technologies” and “Artificial Intelligence Foundations”. Overall information and statistics of all experiments are given in Tab. 4. In the subsequent subsections, we briefly characterize the most important results.


In the first experiment the CKE process in which a team of experts and a team of knowledge engineers work alternately and independently was conducted. Specifically, there were four independent processes led by four two-person expert teams. The first phase was aimed at creation of a wiki about the subject chosen by the team (Pokemons, Simp-
Table 1
Set of assertions for SELECT queries within Loki

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyequal</td>
<td>Attribute name</td>
<td>Checks whether any row / all rows / no rows contain the given value of specified attribute</td>
</tr>
<tr>
<td>allequal</td>
<td>equal, notequal, less equal, less, greater, greater equal</td>
<td>Checks the number of rows returned</td>
</tr>
<tr>
<td>nonequal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2
Set of assertions for ASK queries within Loki

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asserttrue</td>
<td>Empty – one can omit the parameter: [unittest_assert_asserttrue();]</td>
<td>Checks whether the query returns true value</td>
</tr>
<tr>
<td>assertfalse</td>
<td></td>
<td>Checks whether the query returns false value</td>
</tr>
</tbody>
</table>

Table 3
Set of assertions for DESCRIBE queries within Loki

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyequal</td>
<td>Attribute name</td>
<td>Checks whether any / all / no pages contain the given value of specified attribute</td>
</tr>
<tr>
<td>allequal</td>
<td>equal, notequal, less equal, less, greater, greater equal</td>
<td>Checks the number of pages returned</td>
</tr>
<tr>
<td>nonequal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4
Summary of the CKE experiments

<table>
<thead>
<tr>
<th>Duration</th>
<th>Objectives</th>
<th>No. Participants</th>
<th>Changes</th>
<th>Wiki pages</th>
<th>Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Evaluation of the basic CKE process in wiki</td>
<td>8</td>
<td>3000</td>
<td>345</td>
<td>1891</td>
</tr>
<tr>
<td>2nd</td>
<td>Extension of the CKE process based on Ontology 101 [46]</td>
<td>13</td>
<td>1186</td>
<td>265</td>
<td>821</td>
</tr>
<tr>
<td>3rd</td>
<td>Evaluation of Loki features after the 2nd experiment</td>
<td>15</td>
<td>665</td>
<td>202</td>
<td>1031</td>
</tr>
<tr>
<td>4th</td>
<td>Practical comparison of Loki to SMW</td>
<td>105</td>
<td>5046</td>
<td>285</td>
<td>2535</td>
</tr>
<tr>
<td>5th</td>
<td>Evaluation of the modules implementing the agile process</td>
<td>16</td>
<td>579</td>
<td>275</td>
<td>890</td>
</tr>
<tr>
<td>6th</td>
<td>Evaluation of the final version of the process and Loki</td>
<td>105</td>
<td>2807</td>
<td>436</td>
<td>431</td>
</tr>
</tbody>
</table>

Results of this experiment finally led to two conclusions. Firstly, during the CKE process, domain experts and knowledge engineers must collaborate at every stage. Secondly, experiments with students should be conducted under more strictly controlled conditions, not performed occasionally in the students’ free time.

6.2. Second Experiment: CSP Library

In the second experiment students were involved in the project aimed at transferring the knowledge available in Constraint Satisfaction Problem Library
K. Kutt and G.J. Nalepa / Loki – the Semantic Wiki for Collaborative Knowledge Engineering

(15) To the wiki system and preparing relevant semantic annotations. In the experiment, a simple CKE process based on Ontology 101 [46] was implemented. It was done in an iterative manner. Each of 5 iterations lasted 2 hours. During the first iteration, first set of competency questions to CSP Library was developed. After that, ontology prototypes were created.

Subjective teacher observations and analysis of reports prepared by participants led to the following conclusions. First of all, users identified the lack of proper code hint and completion mechanism as a big disadvantage. It was then implemented as described in Sect. 4.6. Secondly, formalization of competency questions to SPARQL queries at the end of the second iteration is too late. In the final version of the CKE process (Sect. 3), this is addressed by incorporation of reasoning unit tests definition (in a form of SPARQL queries), as a part of competency questions step. It was also noted that tasks assignment may be done in a more controlled way. The final version of the CKE process incorporates an iteration board to address this issue. Finally, three roles were observed during the process and implemented later on in the wiki: a leader, a scheme designer, and a developer.

6.3. Third Experiment: Pubs in Cracow

The goal of the experiment was to evaluate new Loki modules in action and to gather feedback about the improved CKE process. The topic of the project was selected by the participants. It was decided to create a wiki about pubs located in the center of Cracow. A new observation was that one person in each team had to be responsible for quality assurance, but in fact such a task may be automated. This was later addressed by introduction of code hint and completion mechanism that also checks the annotations and gives instant feedback of all errors found. Furthermore, the lack of task assignment mechanism was problematic. To address it in a final version of CKE process an iteration planning step that uses an iteration board was introduced.

6.4. Fourth Experiment: Artificial Intelligence Class

In the fourth experiment, students created a wiki about Artificial Intelligence topics covered during the „Artificial Intelligence Foundations” course. Among the main goals of the experiment there were: 1) to verify whether there are differences in difficulty levels of wiki usage between Loki and the popular Semantic MediaWiki (SMW), 2) to test the CKE process conducted by a larger group than in previous experiments. Due to the specific setting, i.e. the fact that the project was done as a part of a course, the process was adjusted with regarding to the original one [13].

To address the first goal, a two-sided Mann-Whitney U test was used to check whether there were any differences between difficulty levels for Loki and SMW reported in survey. This statistical test was selected, as it is designed for determining whether two samples represent different distributions and it does not require the assumption of normal distributions. The test results16 do not allow the null hypothesis of no difference between Loki and SMW to be rejected for each of four measured wiki aspects: an overall wiki usage ($U = 463.0, p = .26$), a wiki text editor ($U = 633.0, p = .20$), semantic annotations ($U = 597.5, p = .46$), and queries ($U = 576.0, p = .46$).

Analysis of process and feedback gathered from users led to the following main observations: 1) a 1 hour introductory training at the beginning was not sufficient for the purposes of the project, 2) groups of 50 people are definitely too big for self-management.

The need for knowledge schemes, proper naming conventions, or even the language used for annotations, was quickly noticed, 3) some users were not convinced of wiki technology. The most frequently pointed problems were related to the UI, and were addressed in the next version of Loki. What is interesting, even with these problems in mind, the vast majority (54 out of 66 votes) decided that the wiki project should be repeated during the next course edition. To conclude, there are no significant differences between reported difficulty for Loki and SMW.

6.5. Fifth Experiment: Cookbook and Movies KB

Once the complete set of Loki modules for the CKE agile process was developed, the fifth experiment was conducted. It was done in two parallel ways: 1) students took part in a CKE agile process done in a way presented in Sect. 3 aimed at cookbook development within Loki, 2) online call for evaluation was announced on Loki web page17 and sent to KE re-

16 See: http://www.csplib.org/.

searchers at AGH-UST, as well as graduates of “Engineering of Intelligent Systems”.
Among the goals of this study there were to a) check whether all modules are compatible with each other, b) conduct an usability study, and c) verify whether the proposed framework improves the CKE process. At the end of the experiment, as well as at the end of tasks connected with call for evaluation, all participants filled the Software Usability Measurement Inventory (SUMI) [64]. As a result of evaluation, the following observations can be made:

1. All modules smoothly interact with each other. No major flaws were noticed (the 1st goal).
2. 18 users filled out the SUMI inventory (see Fig. 10). The results indicates that users generally felt satisfied with the use of the (the 2nd goal).
3. Proposed framework improved the CKE process (the 3rd goal): a) in contrast to previous experiments, no one reported that was “the cop” responsible only for monitoring the quality. This task was supported by the reasoning unit tests module, as well as the hint and completion system. b) thanks to the simple iteration board, everyone knew what tasks are currently being carried out in the wiki. Therefore, there was no redundancy in the work. It was proposed that more advanced iteration boards, like Trello\(^\text{18}\) or JIRA\(^\text{19}\), may be used in the future, e.g. to group tasks.

6.6. Sixth Experiment: Artificial Intelligence Class

Final experiment was a repetition of the fourth one (Sect. 6.4). There were two groups again, with two independent wiki instances: one of them was pure Loki engine, here called “wiki A” (as described in previous papers [6, 54] and summarized in Sect. 4.3) and the second was the new Loki core, “wiki B”, (as presented on Fig. 2 and described in this paper). The goal of this experiment was to investigate how does the new Loki core improve the CKE process in the wiki. Trello was used as a professional iteration board.

Wiki A was used by 58 participants, while wiki B by 47 students. The number of changes made in both wikis was comparable (1540 in wiki A and 1267\(^\text{20}\) in wiki B) but in wiki B there were more wiki pages (257 vs 179) and more triples created (274 vs 157).
It is noticeable that there is a small number of triplets created compared to the number from the fourth experiment (see Tab. 4). The construction of the experiment itself have not changed, but the way of evaluating participation was different. In the 4th experiment, the creation of relationships to other pages was assessed more restrictively, as a result of which participants were “forced” to create multiple triples if they wanted to complete the course with high marks. In the last experiment the rules have been loosened and it was enough to do one triple on a wiki page to “pass”.
All improvements of Loki found in the wiki B were found useful. From the new remarks that emerged after this experiment: The use of Trello as a professional task management tool was very well received. The transparent CKE Agile Process made it clear what steps were being taken. Gamification mechanisms are interesting, but due to focus primarily on passing the course, they were not a motivation for the participants.

7. Related Works and Comparative Evaluation

The first semantic wiki (Platypus) was proposed in 2004. Then, in 2005-2006 “semantic wiki explosion” was observed when many semantic wikis implementations appeared. In subsequent years, more systems were created, but not as many as in the beginning. A comprehensive description of these systems and their history were provided by Bry et al. in 2012 [65]\(^\text{21}\). Only systems that have been updated over the last 5 years (i.e., the latest release or com-

---

\(^\text{18}\)See: https://trello.com/
\(^\text{19}\)See: https://www.atlassian.com/software/jira.

---

\(^\text{20}\)\(^\text{1563}\) when the value is rescaled to the same number of participants

\(^\text{21}\)For list of current and “historical” semantic wikis see also: https://www.semanticweb.org/wiki/Semantic_Wiki_State_Of_The_Art.html and http://www.mkbergman.com/sweet-tools/.
mit were in 2016-2021 period) are considered here: Semantic MediaWiki (Sect. 7.1) (SMW), KnowWE (Sect. 7.2), and OntoWiki (Sect. 7.3). The summary of their features with regard to CKE Requirements (as defined in Sect. 2) is presented in Tab. 5-6. The summary also provides a comparison of these state-of-the-art wikis with Loki. Loki is described separately in two columns: a base Loki engine (as described in Sect. 4.3 and published previously in [6, 54]) is presented in “Loki base” column, a whole updated Loki core (as described in Sect. 4 and previously published only in PhD thesis [23]) is presented in “Current Loki core” column. The WikiMeningitis system [66] is not included as there are no details about the source code or the ability to install the system. As the system is inspired by SMW, it probably has similar capabilities.

7.1. Semantic MediaWiki

Semantic MediaWiki [58, 67] (see Fig. 11) is an extension to the MediaWiki system22. The wiki extends MediaWiki markup to provide a way to describe categories, relations with other wiki pages (object properties) and relations with literal values (data properties):

```
| Category: [[Category:City]]
| Object property: [[Is capital of::United Kingdom]]
| Data property: [[Has population::7,421,329]]
```

Wiki also provides the possibility to: define the ranges of the relations (using categories or data types), state values conversion (e.g. from miles to kilometres), and specify categories and relations hierarchy [58]. There was a notable attempt to create the Distributed SMW system that kept knowledge on multiple servers to be independent of single point of failure [72].

22See: https://www.mediawiki.org/.

7.2. KnowWE

KnowWE (Knowledge Wiki Environment) [63, 68], based on JSPWiki24, is not a simple wiki with se-
<table>
<thead>
<tr>
<th>Publications</th>
<th>SMW</th>
<th>KnowWE</th>
<th>OntoWiki</th>
<th>Loki base</th>
<th>Current Loki core</th>
</tr>
</thead>
</table>

**Agile CKE process**

R1: Agile development is related to the appropriate CKE methodology, not specific tools, so all wikis may be used with CKE process

R2: User’s roles

<table>
<thead>
<tr>
<th></th>
<th>Yes, Unit tests</th>
<th>Yes, Unit tests</th>
<th>Yes, Unit tests</th>
<th>Yes, Unit tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes, Ontology &amp; unit tests</td>
<td>Yes, Ontology &amp; unit tests</td>
<td>Yes, Ontology</td>
<td>No</td>
</tr>
</tbody>
</table>

**Supporting tools**

R4: Compatibility

<table>
<thead>
<tr>
<th></th>
<th>Yes, RDF &amp; SPARQL support</th>
<th>Yes, RDF &amp; SPARQL support</th>
<th>Yes, RDF &amp; SPARQL support</th>
<th>Yes, export to RDF; SPARQL, BPMN &amp; SBVR support; SMW style queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes, RDF &amp; SPARQL support. Compatible with domain-specific standards</td>
<td>Yes, RDF &amp; SPARQL support. Uses well-established vocabularies (SKOS, FOAF, . . .)</td>
<td>Yes, export to RDF; SPARQL, BPMN &amp; SBVR support; SMW style queries</td>
<td>Yes, as in Loki core + ontologies in OWL subset, changelog in PROV</td>
</tr>
</tbody>
</table>

R5: Current KB state

In all Wikis except KnowWE and Loki core there are no automatic tests to validate current state, so the stable and unstable versions are not differentiated. Only KnowWE gives a possibility to download the last stable version of the KB

R6: Adaptation to project

All Wikis have plugin systems that enable the possibility to connect with domain specific tools and adapt the system to the specific project needs

---

*Checked on 10.09.2020.*


*See: http://aksw.org/Projects/Erfurt.html.*
Table 6
Loki core compared to other semantic wikis with regard to the CKE requirements (part 2)

<table>
<thead>
<tr>
<th>Knowledge representation and reasoning</th>
<th>SMW</th>
<th>KnowWE</th>
<th>OntoWiki</th>
<th>Loki base</th>
<th>Current Loki core</th>
</tr>
</thead>
<tbody>
<tr>
<td>R7: Adjustable KRR methods</td>
<td>Yes: media, plain text, annotations, business processes, UML class diagrams</td>
<td>Yes: media, plain text, annotations, production rules, decision trees, decision tables, flowcharts, set-covering models</td>
<td>No, only forms with different types of fields' values</td>
<td>Yes: media, plain text, annotations, Prolog facts and clauses, XTT2 rules, business processes and rules&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Yes, as in Loki core</td>
</tr>
<tr>
<td>Quality management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8: Auto quality checks</td>
<td>No</td>
<td>Yes, unit tests</td>
<td>No</td>
<td>No, only consistency checks limited to XTT2 rules</td>
<td>Yes, as in Loki core + unit tests</td>
</tr>
<tr>
<td>R9: Credibility determination</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes, for users &amp; partially for sources used</td>
</tr>
<tr>
<td>R10: Tools for experts' conflicts</td>
<td>Yes, special page for discussion for each wiki page</td>
<td>No</td>
<td>Yes, discussion about small information chunks</td>
<td>Yes, special page for discussion for each wiki page</td>
<td>Yes, as in Loki core + approval / disagreement mechanism</td>
</tr>
<tr>
<td>Change management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R11: Versioning control</td>
<td>Yes, simple &amp; linear</td>
<td>Yes, simple &amp; linear</td>
<td>Yes, simple &amp; linear</td>
<td>Yes, simple &amp; linear</td>
<td>Yes, robust &amp; graph-based</td>
</tr>
<tr>
<td>User involvement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R12: Usability</td>
<td>Tested in experimental setting, giving good overall results [71]</td>
<td>Not tested</td>
<td>Not tested</td>
<td>Not tested</td>
<td>Tested in experimental setting, giving good overall results [23]</td>
</tr>
<tr>
<td>R13: Gamification</td>
<td>Yes, very simple</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Requirements fulfilled</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>

<sup>a</sup>Processes and rules may be edited and visualized, but they cannot be queried.
It was successfully used in many situations, ranging from historical data in Catalogus Professorum Lipsiensis [78], through medical diagnosis in Dispeidia [17], to animals classification in Caucasian Spiders project [20]. The latter was done in the wild of the Caucasus area, where desktop client is not useful, so there is also a lightweight HTML5-based version of OntoWiki that can be browsed on mobile phones [20].

8. Conclusions

In this paper the field of collaborative knowledge engineering was outlined. To clearly define this concept and differentiate it from related terms, a taxonomy of group-based knowledge engineering was proposed. This allowed for further analysis of CKE and its challenges, which led to the identification of 13 requirements for tools supporting the CKE. Among them, the need for a mature agile methodology for CKE appeared as it has not yet been developed. To address this gap, we have prepared a detailed description of the collaborative knowledge engineering process, which derives both from knowledge engineering methods and from practices developed in the software engineering.

The central part of the paper presents a description of the semantic wiki Loki, a groupware tool that implements the proposed CKE process. Designed as a set of DokuWiki plug-ins, it provides a variety of knowledge representations, including semantic annotations, Prolog clauses, business processes and rules (in BPMN and SBVR notations respectively). It can be easily extended to another ones using plug-ins mechanism. Knowledge stored in Loki can be retrieved via SPARQL queries, inline Semantic MediaWiki-like queries, or Prolog goals. There is also a possibility to export knowledge in RDF/XML files or to query the wiki remotely using SPARQL endpoint. Loki also includes a set of features that facilitate the work of domain experts and knowledge engineers, including knowledge visualization, ontology storage as well as code hint and completion mechanism. The introduction of reasoning unit tests, apart from the method of verifying the quality of knowledge, also facilitates the definition of users’ expectations regarding the final state of the knowledge base. Finally, the description of six experiments conducted to evaluate specific hypotheses during Loki’s and CKE process’ development was provided.

As experiments with students have their limitations, e.g. concerning the motivation of participants, we plan...
to establish cooperation with external partners and use Loki for real use case related to digital humanities or software engineering project. The latter will be possible thanks to the ability to store knowledge in the form of business processes and rules, and the plug-in system that facilitates the expansion with new knowledge representations. We consider this to be a solid basis for storing software engineering project specifications in the wiki. Unlike simple wiki systems hosted in Gitlab or Azure DevOps, Loki will allow better processing and querying of stored knowledge, as well as verification of its consistency. A connector between the programming IDE and the wiki, allowing users to link specific pieces of code and fragments of specification stored in the wiki is under development. The current development is also aimed at improving human-wiki interaction through the gamification and adaptive UI modules (see Fig. 2). Finally, optimization of the engine itself is also planned. The most important one is to introduce the possibility of storing knowledge in a native triplestore, which will increase the efficiency of the system for larger knowledge bases.

References


