WikiApp – Engineering of Domain-specific Wiki Applications

Darya Tarasowa, Ali Khalili, Ivan Ermilov, Sören Auer

Universität Leipzig, II/BI/S/AKSW, Johannisgasse 26, 04103 Leipzig, Germany
{lastname}@informatik.uni-leipzig.de, http://aksw.org

Abstract. Since its inception in the early 2000s, wiki technology became a ubiquitous pillar for enabling large-scale collaboration. However, the wiki paradigm was mainly applied to unstructured, textual content thus limiting the content structuring, repurposing and reuse. More recently with the appearance of semantic wiki’s the concept was also applied and extended towards semantic content with adverse effects on scalability. Often, however, (semi-)structured content should be managed and the collaboration of potentially very large user communities around such content should be effectively facilitated. In this paper we present a model-driven approach for applying the wiki paradigm to semi-structured content. We present a data model and a model-driven generation approach to implement it. Structured content objects are versioned thus mimicking sophisticated versioning control. We implement and evaluate our model with SlideWiki – an example Web application facilitating the collaboration around educational content. The article also comprises a performance and scalability evaluation using synthetic benchmarking.

1 Introduction

Since its inception in the early 2000s [1], wiki technology became a ubiquitous pillar for enabling large-scale collaboration. Traditional, text-oriented wikis enabled the creation of the largest encyclopedia of human-mankind edited by tens of thousands of volunteer editors – Wikipedia. Also, for group collaboration, in corporate intranets or online communities wiki’s meanwhile constitute a fundamental base technology. However, Ward Cunningham’s wiki paradigm was mainly only applied to unstructured, textual content thus limiting the content structuring, repurposing and reuse. More recently with the appearance of semantic wiki’s the concept was also applied and extended to semantic content. Two kinds of semantic wikis exist: Semantic text wikis, such as Semantic MediaWiki [2] are based on semantic annotations of the textual content. Semantic data wikis, such as OntoWiki [3], are directly based on the RDF data model. In many potential usage scenarios, however, the content to be managed by a wiki is neither purely textual nor fully semantic. Often (semi-)structured content (e.g. presentations, educational content, laws, skill profiles etc.) should be managed. The collaboration of large user communities around such content should be effectively facilitated. Also, semantic wiki applications suffer from low scalability,
that restricts usage of this technology. In this paper we present the *WikiApp data model* for applying the wiki paradigm to semi-structured content and the *model-driven approach* to generate domain-specific applications.

The WikiApp data model is a refinement of traditional entity-relationship data models. There are two relations in the heart of the model: *part-of* relation to structure the content and *based-on* relation to support the creation of a collaboration eco-system. Employing the single reflexive *based-on* relation content objects are versioned thus mimicking sophisticated versioning control including branching and merging. In essence, the WikiApp data model adds additional formalisms in order to make users as well as ownership, part-of, based-on and following relationships first-class citizens of the data model. The model could be also viewed as an example of the design paradigm, known as the *matryoshka principle*\(^1\). Similar to the nested doll, a set of content objects connected by *part-of* relations can be arranged and manipulated in exactly the same manner, as an individual non-structured object. The WikiApp data model directly supports social networking features, such as following of other users and content objects. Furthermore, the proposed data model allows the direct publishing of RDF as Linked Data.

We implement, showcase and evaluate our model with the SlideWiki application – an example Web application facilitating the collaboration around educational content. In particular we make the following contributions:

- We propose the WikiApp data model for the engineering of (semi)-structured content in a truly collaborative way. We describe base operations on this data model and procedures for comprehensive revision management.
- We present the model-driven *Wikifier Code Generator* and an example domain-specific language, developed to support the rapid generation of WikiApp applications.
- With our example implementation SlideWiki, we showcase two comprehensive examples of using the model proving its flexibility and perform a scalability evaluation.

This article is structured as follows: We present the concepts underlying our data model in **Section 2**. Our model-driven approach for the rapid development of WikiApp implementations is presented in **Section 3**. We discuss our implementation in **Section 4**. An evaluation is provided in **Section 5**. We review related work in **Section 6** and conclude with an outlook on future work in **Section 7**.

### 2 Concept

In this section we introduce the fundamental WikiApp concepts. It is based on the following principles:

- **Social collaboration.** Following other users, watching the evolution of content as well as reusing and re-purposing of content in social collaboration networks is at the heart of the WikiApp model.

\(^1\) [http://en.wikipedia.org/wiki/Matryoshka#As_metaphor](http://en.wikipedia.org/wiki/Matryoshka#As_metaphor)
Simplicity. WikiApp implementations should be simple to build and use.

Transparency, openness and peer-review. Content in a WikiApp implementation should be visible and easily observable for the largest possible audience, thus facilitating review and quality improvements.

Provenance. The origin and creation context of all information in a WikiApp implementation should be preserved and well documented.

Scalability. WikiApp implementations should be scalable and implementable according to established Web application development practices (such as the MVC pattern).

The aim of the WikiApp concept is to provide a framework for implementing these principles similarly to Ward Cunningham’s wiki concept [1] for traditional text wikis. However, due to the increased complexity of the (semi-)structured content, just a high level description of the principles is not sufficient to support the creation of domain-specific WikiApp implementations. By devising a formal concept we aim to provide a clear and consistent description of the approach. This simplifies the creation of concrete instantiations and can be used as a basis for integrating WikiApp support into engineering methodologies, development frameworks as well as model-driven code generators. In the sequel, we present a formal description of the WikiApp data model and describe the base operations on this data model.

2.1 Data Model

The WikiApp data model is a refinement of the traditional entity-relationship data model. It adds some additional formalisms in order to make users as well as ownership, part-of and based-on relationships first-class citizens of the data model. The UML meta-model is available from [http://aksw.org/projects/Slidewiki](http://aksw.org/projects/Slidewiki). We illustrate the model in Figure 1 and formally define as follows:

**Definition 1 (WikiApp data model).** The WikiApp data model WA can be formally described by a triple $\mathcal{M} = (U, T, O)$ with:

- $U$ - a set of users.
- $T$ - a set of content types with associated property types $P_t$ having this content type as their domain.
- $O = \{O_{t} \in T\}$ with $O_t$ being sets of content objects for each content type $t \in T$.

Each $O_t$ consists of content objects $o_{t,i}$ with $i \in I_T$ being a suitable identifier set for the content objects in $O_t$. Each $o_{t,i}$ comprises a set of properties $P_{t,i} = Attr_{t,i} \cup Rel_{t,i}$. $Attr_{t,i}$ is a set of literal, possibly typed attributes, and $Rel_{t,i}$ is a set of relationships with other content objects; The only necessary attribute for all content objects is $c_{t,i}$, which contains the creation timestamp of the object $o_{t,i}$. $Rel_{t,i}$ can particularly include the following designated relationships to related objects:

- $part_{t,i} \subset O$ refers to set of the content objects contained in $o_{t,i}$.
– base\(_{t,i}\) \(\in\) \(O_t\) refers to base content object from which the object \(o_{t,i}\) was derived;
– user\(_{t,i}\) \(\in\) \(U\) refers to a user being the owner of the \(o_{t,i}\);

The WikiApp model assumes that all content objects are versioned using the timestamp \(c_{t,i}\) and the base content object relation \(b_{t,i}\). In practice, however, usually only a subset of the content objects are required to be versioned. For auxiliary content (such as user profiles, preferences etc.) it is usually sufficient to omit a base content object relation. For reasons of simplicity of the presentation and space restrictions we have omitted a separate consideration of such content here. However, this is in fact just a special case of our general model, where the base content object relation \(base_{t,i}\) is empty for a subset of the content objects.

The WikiApp data model is compatible with both the relational data model and the RDF data model (i.e. it is straightforward to map it to each one of these). When implemented as a relational data, content types correspond to tables and content objects to rows in these tables. Functional attributes and relationships as well as the owner and base-content-object relationships can be modeled as columns (the latter three representing foreign-key relationships) in these tables. For \(1-n\) and \(m-n\) relationships and non-functional attributes suitable helper tables have to be created. The implementation of the WikiApp model in RDF is slightly more straightforward: content types resemble classes and content objects instances of these classes. Attributes and relationships can be attached to the classes via rdfs:domain and rdfs:range definitions and directly used as properties of the respective instances. For reasons of scalability we expect the WikiApp data model to be mainly used with relational backends. However, using techniques such as Triplify [4] or other RDB2RDF mapping techniques a
Linked Data interface can be easily added to any WikiApp implementation (cf. Section 4.4).

2.2 Operations

After we introduced the data model, we now describe the main operations on the data model.

**Definition 2 (WikiApp operations).** Base operations defined on the WikiApp model are:

- \( \text{create}(u, t, p) : U \times T \times P_t \rightarrow O_t \) creates a new content object of type \( t \) with the owner \( u \) and properties \( p \).
- \( \text{newRevision}(u, t, i, p) : U \times T \times I_T \times P_t \rightarrow O_t \) creates a copy of an existing content object \( o_{t,i} \) of type \( t \) potentially with a new owner \( u \) and overriding existing properties with \( p \).
- \( \text{getRevision}(t, i) : T \times I_T \rightarrow O_t \cup \emptyset \) returns the existing content object \( o_{t,i} \) of type \( t \) including all its properties or \( \emptyset \) in case a content object of type \( t \) with identifier \( i \) does not exist.
- \( \text{isWatching}(u, t, i) : U \times T \times I_T \rightarrow \{\text{true, false}\} \) returns true if the user \( u \) is watching the content object of type \( t \) with identifier \( i \) or false otherwise. Following users is a special case, where the content object type is set to user.
- \( \text{watch}(u, t, i) : U \times T \times I_T \rightarrow \{\text{true, false}\} \) toggles user \( u \) watching the content object of type \( t \) with identifier \( i \) and returns the new watch status.

In the spirit of the wiki paradigm, there is no deletion or updating of existing, versioned content objects. Instead new revisions of the content objects are created and linked to their base objects via the \( \text{base}_{t,i} \) relation. All operations have to be performed by a specific user and the newly created content objects will have this user being associated as their owner. In addition, when a new revision of a content object is created and the original content object is indicated to be part of another object, the creation of a new revision of the containing object has to be triggered as well. In our implementation (cf. Section 4.1), this is, for example, triggered when a user creates a new revision of a slide being part of a deck. If the user is not the owner of the containing deck, a new deck revision is automatically created not to implicitly modify other users’ decks (cf. Section 4.3). Watching the content, as well as following the users, allows users to receive the information about changes of the followed content object or new objects created by the watched user. Also, these operations allow to easily find the followed object or user. The flexibility of the model allows to add additional operations.

3 Model-driven generation of WikiApp implementations

Using a model-driven Web application engineering approach, developers are able to easily and quickly implement WikiApp applications. We devised a Domain-
Specific Language (DSL) based on the WikiApp Data Model and a transformation approach implemented in a tool called Wikifier\(^2\) which receives a WikiApp definition in the DSL and generates the appropriate database (or RDF) schema, classes and methods for interacting with this model as well as the required SQL (or SPARQL) queries. The Wikifier DSL is dedicated to the specific WikiApp problem representation technique. In essence its a YAML-formatted\(^3\) file with the definition of content types, their attributes, relations and part-of relations according to the WikiApp data model (cf. Definition\(^1\)). We chose YAML format because it is more human readable than XML and is easily convertible to XML. Figure\(^2\) shows an instantiation of this DSL for our example application.

The Wikifier model transformation is integrated into the code generator of the Symfony framework\(^4\) which is based on the Model View Controller (MVC) design pattern. Symfony is a popular open-source Web application framework with a large community of users thereby using it will lower the entrance barrier of WikiApp users. Symfony code generator transforms the DSL instantiation to the corresponding data models with basic CRUD operations and the corresponding views and controllers. The generated models will include the following extensions derived from the WikiApp data model:

(a) A revision model for each content object with timestamp and based_on properties.
(b) A partOf model which includes the identifier properties of the selected revision models.
(c) A subscription model which is used for following revision models.
(d) A user model which is referred by each of the generated revision models.

In addition to the generic WikiApp operations (cf. Definition\(^2\)) Wikifier creates convenience methods for performing these operations directly from the respective content object classes.

4 WikiApp Example Implementation

In this section we describe SlideWiki - concrete WikiApp implementation, which we created to demonstrate the effectiveness and efficiency of our model\(^5\). The main idea of SlideWiki is to enable the crowd-sourcing of educational content, in particular presentations and questionnaires. The data models introduced below show the relatively complex relationships between objects.

4.1 Implementation of WikiApp data model in SlideWiki

Our SlideWiki example application uses two implementations of WikiApp data model.

\(^2\) Available at: [http://slidewiki.aksw.org/wikifier/](http://slidewiki.aksw.org/wikifier/)

\(^3\) [http://yaml.org/](http://yaml.org/)

\(^4\) [http://symfony.com/](http://symfony.com/)

\(^5\) SlideWiki is available at [http://slidewiki.aksw.org](http://slidewiki.aksw.org) with a accompanying video screencast at: [http://slidewiki.aksw.org/video/slidewiki.mp4](http://slidewiki.aksw.org/video/slidewiki.mp4)
The first implementation (Example 1) is used for managing slides and presentations. It includes individual slides (consisting mainly of HTML snippets, SVG images and meta-data), decks (being ordered sequences of slides and sub-decks), themes (which are associated as default styles with decks and users) and media assets (which are used within slides).

The second implementation (Example 2) was developed for managing questions and questionnaires (tests). It includes questions for the slide material (the question is assigned to all slide revisions), questionnaires (which could be organized manually by user or created automatically in accordance with the deck content), and answers (which are the part of the questions).

Because of the space limitations, we do not describe here the attributes and relations that are necessary for all the content types (e.g. timestamp attribute, is_owner and based_on relationship). Also, for the better structuring, we use here the opposite has_part relation instead of is_part.

Example 1 (The data model instance for slide presentations).

- \( T = \{\text{deck, slide, theme, media}\} \)
- \( \text{Attr}_{\text{deck}} = \{\text{title} \rightarrow \text{text}, \text{abstract} \rightarrow \text{text}, \)
  \( \text{license} \rightarrow \{"CC-BY","CC-BY-SA"\}, \)
  \( \text{default_theme} \rightarrow \text{theme}\}, \)
- \( \text{Rel}_{\text{deck}} = \{\text{has_part} \rightarrow \{\text{deck, slide}\}\} \)
- \( \text{Attr}_{\text{slide}} = \{\text{content} \rightarrow \text{text}, \text{speaker_note} \rightarrow \text{text}, \)
  \( \text{license} \rightarrow \{"CC-BY","CC-BY-SA"\}\}, \)
- \( \text{Rel}_{\text{slide}} = \{\text{has_part} \rightarrow \text{media}\}, \)
- \( \text{Attr}_{\text{media}} = \{\text{type} \rightarrow \{\text{image, video, audio}\}, \)
  \( \text{uri} \rightarrow \text{string}, \)
  \( \text{license} \rightarrow \{"CC-BY","CC-BY-SA"\}\}, \)
- \( \text{Rel}_{\text{theme}} = \{\}\)  

Example 2 (The data model instance for questionnaires).

- \( T = \{\text{test, question, answer}\} \)
- \( \text{Attr}_{\text{test}} = \{\text{type} \rightarrow \{"manual","auto"\}\}, \)
- \( \text{Rel}_{\text{test}} = \{\text{has_part} \rightarrow \{\text{question, test}\}\} \)
- \( \text{Attr}_{\text{question}} = \{\text{question} \rightarrow \text{text}, \text{difficulty} \rightarrow \text{Real}\}, \)
- \( \text{Rel}_{\text{question}} = \{\text{has_part} \rightarrow \text{answer}\}, \)
- \( \text{Attr}_{\text{answer}} = \{\text{answer} \rightarrow \text{text}, \text{is_correct} \rightarrow \text{boolean}, \text{explanation} \rightarrow \text{text}\}\)  
- \( \text{Rel}_{\text{answer}} = \{\}\)

4.2 DSL model

After the data models were build, we described them, using the YAML syntax. An example of such descriptions is present in Figure 2.
4.3 Revision control

Although substantially simplified by the WikiApp data model revision control is an important issue in SlideWiki. In accordance with the WikiApp data model, content objects are never updated or deleted, instead new revisions will be created. As shown in Figure 3, there are different cases for which new object revisions have to be created.

For decks, however, the situation is slightly more complicated, since we wanted to avoid an uncontrolled proliferation of deck revisions. This would, however, happen due to the fact, that every change of a slide would also trigger the creation of a new deck revision for all the decks the slide is a part of. Hence, we follow a more retentive strategy. We identified three situations which have to cause the creation of new revisions:

- The user specifically requests to create a new deck revision.
- The content of a deck is modified (e.g., slide order is changed, change in slides content, adding or deleting slides to/from the deck, replacing a deck content with new content, etc.) by a user other than the owner of a deck.
- The content of a deck is modified by the owner of a deck but the deck is used somewhere else.
In addition, when creating a new deck revision, we always need to recursively spread the change into the parent decks and create new revisions for them if necessary.

As it was already mentioned, there are two different types of questionnaires. The automatically created questionnaires are based on the current deck revision structure. When a new revision of the question is created, it automatically appears in the questionnaire. Thus questionnaires do not need version control. Manually created questionnaires present a collection of chosen questions and can not be manipulated as objects. Thus, in our implementation only questions and answers have to be placed under the version control. However, their structure is trivial and the logic of creating their new revisions is intuitive. We just restricted the number of new revisions to be created similarly with the decks: changes made by the question owner do not trigger a new revision creation.

4.4 Linked Data Interface

WikiApp implementations can be easily equipped with a Linked Data interface. We employed the RDB2RDF mapping tool Triplify [4] to map SlideWiki content to RDF and publish the resulting data on the Data Web. Triplify is based on mapping HTTP-URI requests onto relational database queries. It transforms the resulting relations into RDF statements and publishes the data on the Web in various RDF serializations, in particular as Linked Data. Triplify neither defines nor requires to use a new mapping language, but exploits and extends certain SQL notions with suitable conventions for transforming database query results (or views) into RDF and Linked Data. The Triplify configuration for SlideWiki was created manually, but we envision that this can be in future performed automatically from respective WikiApp DSL representation. The SlideWiki Triplify Linked Data interface is available via: [http://slidewiki.aksw.org/triplify](http://slidewiki.aksw.org/triplify)
5 Evaluation

With the evaluation we aim to verify our claim, that WikiApp implementations will scale better than comparable semantic wiki applications with a synthetic benchmark of the SlideWiki WikiApp implementation.

We synthesized three datasets containing 100, 500 and 1000 presentations in pptx format, with an average of 33 slides in each presentation. The presentations were obtained from the two websites: 2shared.com (900 presentations) and slideshare.com (100 presentations). We randomly chose presentations with a file size between 3MB and 20MB, in order to exclude empty and excessively large presentations. A notebook computer with Intel Core 2 Duo 2.66GHz CPU; 4GB RAM and Windows 7 64-bit was used for all the measurements. One of the advantages of a relational database in comparison with a triple-store is better performance and scalability. To demonstrate this, we imported an RDF dump produced by Triplify (cf. Section 4.4), into OntoWiki [3] (with Virtuoso triple-store backend), and repeated the measurements with corresponding SPARQL queries. To compare the scalability of SlideWiki, the average execution time for each operation in the relational and RDF implementations was plotted relatively to the 100-presentations dataset in Figure 4. The results of these measurements are presented in the Figure 4. We also plotted the average execution time relatively to the 100-deck dataset in the right part of Figure 4. Based on the measurement results, we can conclude, that SlideWiki performs better in all operations, other than move deck and new deck revision. SlideWiki also scales significantly better, despite the fact, that we did not yet invest much effort in optimizing the performance. In particular, we will further optimize the performance of the least scaling operations show deck and search slide.

6 Related Work

Related work can be roughly divided into wiki-based knowledge engineering, semantic wikis and model-driven web engineering. Wiki-based Collaborative Knowledge Engineering The importance of wikis for collaborative knowledge engineering is meanwhile widely acknowledged. In [5], for example, a knowledge engineering approach which offers wiki-style collaboration is introduced aiming to facilitate the capture of knowledge-in-action which spans both explicit and tacit knowledge types. The approach extends a combined rule and case-based knowledge acquisition technique known as Multiple Classification Ripple Down Rules to allow multiple users to collaboratively view, define and refine a knowledge base over time and space. In a more applied context, [6] introduces the concept of wiki templates that allow end-users to define the structure and appearance of a wiki page in order to facilitate the authoring of structured wiki pages. Similarly the Hybrid Wiki approach [7] aims to solve the problem of using (semi-)structured data in wikis by means of page attributes. WikiApp differs from such general purpose wiki-based knowledge engineering methodologies due to its model-driven generation of domain-specific
Fig. 4. Benchmarking results: (left) Full comparison (right) Execution time relatively to 100-deck dataset. Legend: 1-show deck, 2-search slides, 3-new slide revision, 4-move slide, 5-new deck revision, 6-move deck, 7-search deck.

Semantic wikis Another approach to combine wiki technology with structured representation are semantic wikis [8]. There are two types of semantic wikis. Semantic text wikis, such as Semantic MediaWiki [2] or KiWi [9] are based on semantic annotations of the textual content. Semantic data wikis, such as OntoWiki [3], are based on the RDF data model in the first place. Both types of semantic wikis, however, suffer from two disadvantages. Firstly, their performance and scalability is restricted by current triple store technology, which is still an order of magnitude slower when compared with relational data management, which is regularly confirmed by SPARQL benchmarks such as BSBM [10]. Secondly, semantic wikis are generic tools, which are not particularly adapted for a certain domain thus substantially increase the usage complexity for users. The latter problem was partially addressed by OntoWiki components such as

6 http://trac.adhocracy.cc/
Erfurt API, RDFauthor and Semantic Pingback, which evolved OntoWiki into a framework for Web Application development [3].

**Model-driven web engineering** There are a number of approaches, which applied the model-driven paradigm to the development of web applications. A first category of work such as Autoweb [11] or more recently WebForm Diagram [12] aimed at improving the Web content management by focusing on traditional Web content structures, such as Web pages, their meta-data, navigation, linking and forms. The model-driven development of sophisticated Web application was pioneered by WebML and its CASE tool WebRatio [13]. WebML [14] defines four fundamental models (structural, hypertext, presentation and personalization) but uses classical notations like E/R, object-oriented and UML models as plugins for the structural description of a Web application. Our WikiApp concept in turn, focuses on refining the structural description with additional data structures for enabling wiki-based Web applications. Regarding other aspects, such as presentation, linking etc. WikiApp can be easily combined with WebML-like models or modern Web application development techniques. Similar in spirit to WebML, but tighter aligned with UML is the UML-based Web Engineering approach (UWE) with its generator UWE4JSF [15]. There are a number of recent advances with regard to integration of specific aspects into model-driven Web application engineering, such as evolution and adding of concerns [16]; requirements and user-interface-driven development [17] or usability [18]. We deem WikiApp due to its focus on the data structures being orthogonal to most of these approaches and thus being easily combinable.

With WikiApp we aim to find a sweet spot on the balance between functionality and complexity. For example, by using the simple YAML format for data transformations, WikiApp substantially decreases the entrance barrier for developers compared to other approaches, since knowledge of RDF, RDFS and OWL is not required. Also, our approach provides a set of important features for wiki-like applications, that are not available in other frameworks, such as version control and crowdsourcing (cf. Table 1). Thereby we make a trade-off between complexity and set of features, maintaining only useful for wiki-like applications features and at the same time keeping the overall framework relatively simple.

**7 Conclusions and further work**

In this paper we presented the WikiApp data model and accompanying tools for creating domain-specific wiki-like applications. Our approach addresses weaknesses of conventional text-oriented as well as semantic wikis. Compared to the text-oriented wikis, WikiApp implementations allow a structuring of the content, resulting in substantially increased reuse, repurposing and collaboration capabilities. With regard to semantic wikis, WikiApp implementations are more user friendly (since they can be tightly adapted to the respective domain) and scalable (since they can be used in conjunction with relational databases).
We see this effort as the first step in a larger research and engineering agenda. We expect the WikiApp model to be applied to many more application domains beyond our current application for educational content with SlideWiki.

As a further work we plan to increase the support of the model-driven WikiApp implementations generation. We plan extensions to the WikiApp DSL and the Wikifier transformation, which enable the automatic creation of user interface components and the use of cloud computing infrastructure.

Another direction of the future research is a support of multi-linguality. Since all content is versioned and semantically structured, it is easier to (automatically) translate content and to keep track of changes in various multi-lingual versions of the same content object by comparing and merging the versions. We want WikiApp data model to have native ability to support such functions.

References


