WikiApp – Engineering of Domain-specific Wiki Applications

Sören Auer
Technische Universität Chemnitz
soeren.auer@informatik.tu-chemnitz.de

Ali Khalili
Universität Leipzig, AKSW
khalili@informatik.uni-leipzig.de

Darya Tarasowa
Universität Leipzig, AKSW
darya.tarasowa@gmail.com

Ivan Ermilov
Universität Leipzig, AKSW
ivan.ermilov@informatik.uni-leipzig.de

Timofey Ermilov
Universität Leipzig, AKSW
ermilov@informatik.uni-leipzig.de

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 Wiki 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 and for the engineering of domain-specific wiki applications. The approach is based on the definition of content types, objects and their relationships. Based on a single reflexive relation content objects are versioned thus mimicking sophisticated versioning control. We implement and evaluate our approach with SlideWiki – a Web application facilitating the collaboration around educational content. With SlideWiki users can author, collaborate and arrange slides in presentations. Presentations can be organized hierarchically, so as to structure them reasonably according to their content. The article also comprises an evaluation using synthetic benchmarking as well as involving real users.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; K.3.1 [**Computer Uses in Education**]: Computer-assisted instruction (CAI); D.2.10 [**Software Engineering**]: Design

General Terms

Design, Standardization, Human Factors

1. INTRODUCTION

Since its inception in the early 2000s [13], 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 [11] are based on semantic annotations of the textual content. Semantic data wikis, such as OntoWiki [9], 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 and the collaboration of *large* user communities around such content should be effectively facilitated.

In this paper we present the *WikiApp* approach for applying the wiki paradigm to semi-structured content. The approach is based on the definition of content types, content objects and their relationships in the WikiApp data model. The WikiApp data model is a refinement of traditional entity-relationship data models. It adds some 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. Employing the single reflexive *based-on* relation content objects are versioned thus mimicking sophisticated versioning control including branching and merging as we know it from software sourcecode versioning systems such as Subversion, Git and Mercurial. In the spirit of the wiki paradigm, there is no deletion or updating of existing content objects. Instead new revisions of content objects are created and possibly linked to their base objects via the *based-on* relation. The WikiApp data model also directly supports *social networking features*, such as *following* of other users and the *watching* of content. The notification of users watching certain content objects is di-

rectly facilitated through the data model by specifically designating *part-of* relations between content objects. For the creation of concrete WikiApp implementations we developed a domain-specific language and the *model-driven* generation approach *Wikifier*, which takes a WikiApp data model as input and generates a comprehensive object model for *rapid application development*. Furthermore, the WikiApp data model also allows the direct publishing of RDF as Linked Data.

We implement, showcase and evaluate our approach with the SlideWiki application – a Web application facilitating the collaboration around educational content. With SlideWiki users can create and collaborate on slides and arrange slides in presentations. Presentations can be organized hierarchically, so as to structure them reasonably according to their content. We have chosen the educational domain, since we deem an implementation of the WikiApp approach here to have the highest impact. Currently large-scale collaboration (also refereed to as *crowd-sourcing*) around educational content (other than texts[1]) is supported only in a very limited way. Slides, presentations, diagrams, assessment tests etc. are mainly created by tutors, teachers, lecturers and professors individually or in very small groups. The resulting content can be shared online (examples include *Slideshare.net*, *OpenStudy.com*, *Google Docs*). However, a proper community collaboration, authoring, versioning, branching, reuse and re-purposing of educational content similarly as we know it from the open-source software community is currently not supported.

With SlideWiki we showcase an implementation of the WikiApp concept, where potentially large communities of teachers, lecturers, academics are empowered to create sophisticated educational content in a truly collaborative way. For newly emerging research fields, for example, a collaboration facility such as SlideWiki allows to disseminate content and educate PhD students and peer-researchers more rapidly, since the burden of creating and structuring the new field can be distributed among a large community. Specialists for individual aspects of the new field can focus on creating educational content in their particular area of expertise and still this content can be easily integrated with other content, re-structured and re-purposed. A particular aspect, which is facilitated by SlideWiki (and WikiApp implementations in general) is multi-linguality. Since all content is versioned and semantically structured, it is trivial to (automatically) translate content and to keep track of changes in various multi-lingual versions of the same content object. As a consequence, we expect that the SlideWiki approach will have a substantial impact with regard to disseminating educational content in many languages.

In particular we make the following contributions:

- We develop the WikiApp approach for the engineering of domain-specific wiki applications based on a specifically tailored data model, operations on this data model and comprehensive revision control. The rapid WikiApp application development is supported by the model-driven Wikifier code generator.
- With SlideWiki, we showcase a comprehensive example application in the collaboration around educational content domain.
- In our evaluation we demonstrate, that despite its complex content types and comprehensive content versioning sufficient scalability can be achieved and due to the domain-specificity the complexity of the data model can be effectively hidden from end users.

[1]Wikibooks.org is an example for large-scale collaboration around textual educational content.

This article is structured as follows: We present the concepts underlying our WikiApp approach in Section 2. Our model-driven approach for the rapid development of WikiApp implementations is presented in Section 3. We discuss our implementation including crucial functionality such as versioning, Linked Data interface and search in Section 4. An evaluation using synthetic benchmarking as well as involving real users is provided in Section 5. We review related work in Section 6 and conclude with an outlook on future work in Section 7.

2. CONCEPTS

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

- *Provenance.* The origin and creation context of all information in a WikiApp implementation should be preserved and well documented.
- *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.
- *Simplicity.* WikiApp implementations should be simple to build and use.
- *Social collaboration.* Following other users, watching the evolution of content as well as reusing and repurposing of content in social collaboration networks is at the heart of WikiApp.
- *Scalability.* WikiApp implementations should be scalable and be 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 [13] for traditional text wikis. However, due to the increased complexity of the (semi-)structured content and operations on this content just a high level description of principles is not sufficient to support the creation of domain-specific WikiApp implementations. By devising a formal WikiApp concept we aim to provide a clear and consistent description of the approach, which simplifies the creation of concrete WikiApp instantiations and can be used as a basis for integration 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 then the base operations on this data model.

2.1 Data Model

The WikiApp data model is a refinement of traditional entity-relationship data model. It adds some additional formalisms in order to make users as well as ownership, part-of and derived-from relationships first-class citizens of the data model.

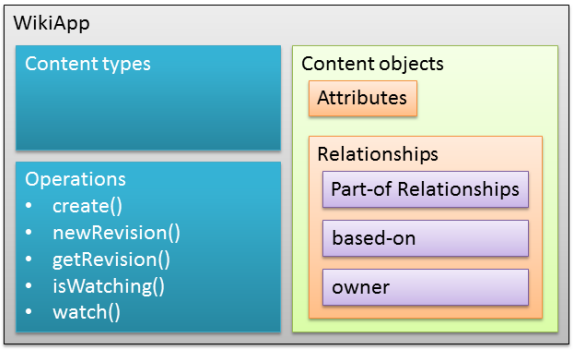

Figure 1: Schematic view of the WikiApp data model.

Definition 1 (WikiApp data model). *The WikiApp data model $\mathcal{WA}$ can be formally described by a triple $\mathcal{WA} = (U, T, O)$ 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 $o_{t,i} = \{P_{t,i}, b_{t,i}, u_{t,i}, c_{t,i}\}$ with:*
  - *$i \in I_T$ being a suitable* identifier *set for the content objects in O_t;*
  - *properties $P_{t,i} = Attr_{t,i} \cup Rel_{t,i} \cup Part_{t,i}$ with $Attr_{t,i}$ being a set of literal, possibly typed* attributes*, $Rel_{t,i}$ being a set of* relationships *with other content objects, $Part_{t,i}$ being a set of* part-of and has-part relationships *referring to other content objects;*
  - *$b_{t,i} \in O_t \cup NULL$ referring to* base content object *from which this content object was derived;*
  - *$u_{t,i} \in U$ referring to a user being the* owner *of this content object;*
  - *$c_{t,i}$ containing the creation timestamp of object $o_{t,i}$.*

The WikiApp data 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 the general WikiApp data model, where the base content object relation $b_{t,i}$ is empty for a subset of the content objects.

The WikiApp data model is compatible with *both* the relational data model as well as the RDF data model. When implemented as 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 data 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 [1] or other RDB2RDF [21] mapping techniques a Linked Data interface can be easily added to any WikiApp implementation (cf. Section 4.3).

Example 1 (SlideWiki data model). *For our SlideWiki example application (whose implementation is explained in detail in Section 4) the data model consists of individual slides (consisting mainly of HTML snippets and some meta-data), decks (being ordered sequences of slides and sub-decks), media assets (which are used within slides) as well as themes (which are associated as default styles with decks and users):*

- $T = \{deck, slide, media, theme\}$
- $Attr_{deck} = \{title \rightarrow text, abstract \rightarrow text, license \rightarrow \{CC-BY, CC-BY-SA\}\}$, $Rel_{deck} = \{default\_theme \rightarrow theme\}$, $Part_{deck} = \{deck\_content \rightarrow deck \cup slide\}$
- $Attr_{slide} = \{content \rightarrow text, speaker\_note \rightarrow text, license \rightarrow \{CC-BY, CC-BY-SA\}\}$, $Rel_{slide} = \{uses \rightarrow media\}$, $Part_{slide} = \{\}$
- $Attr_{media} = \{type \rightarrow \{image, video, audio\}, uri \rightarrow string, license \rightarrow \{CC-BY, CC-BY-SA\}\}$, $Rel_{media} = \{\}$, $Part_{media} = \{\}$
- $Attr_{theme} = \{title \rightarrow string, css\_definition \rightarrow text\}$, $Rel_{theme} = \{\}$, $Part_{theme} = \{\}$

2.2 Operations

After we introduced the WikiApp data model, we now describe the main operations on the data model. In the spirit of the Wiki paradigm, there is no deletion or updating of existing, versioned content objects. Instead new revisions of content objects are created and linked to their base objects via the $b_{t,i}$ relation.

Definition 2 (WikiApp operations). *Five base operations are defined on the WikiApp data model:*

- $create(u, t, p) : U \times T \times P_t \rightarrow O_t$ *creates a new content object of type* t *with the owner* u *and properties* p.
- $newRevision(u, t, i, p) : U \times T \times I_T \times P_t \rightarrow O_t$ *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.
- $getRevision(t, i) : T \times I_T \rightarrow O_t \cup false$ *returns the existing content object* $o_{t,i}$ *of type* t *including all its properties or false in case a content object of type* t *with identifier* i *does not exist.*
- $isWatching(u, t, i) : U \times T \times I_T \rightarrow \{true, false\}$ *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.*
- $watch(u, t, i) : U \times T \times I_T \rightarrow \{true, false\}$ *toggles user* u *watching the content object of type* t *with identifier* i *and returns the new watch status.*

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 an existing content object is created and the original content object is indicated to be part of another content object (by the distinguished part-of relations) the creation of a new revision of the containing content object has to be triggered as well. In our Example 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, so as to not implicitly modify other users' decks (cf. Section 4.2).

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). Figure 2 shows an instantiation of this DSL for our SlideWiki 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. It transforms the DSL instantiation to the corresponding data models with basic *CRUD operations* and the corresponding

[2]Available at: `http://slidewiki.aksw.org/wikifier/`
[3]`http://yaml.org/`
[4]`http://symfony.com/`

```
Deck:
  attributes:
    title: {type:string}
    abstract: {type:text}
    license: {type:enum,values:['CC-BY','CC-BY-SA']}
  relations:
    Theme:{name:default_theme}
  parts:
    Deck: ~
    Slide: ~
Slide:
  attributes:
    content: {type:text}
    speaker_note: {type:text}
    license: {type:enum,values:['CC-BY','CC-BY-SA']}
  relations:
    Media:{name:uses}
Media:
  attributes:
    type: {type:enum,values:['image','video','audio']}
    uri: {type:string}
    license: {type:enum,values:['CC-BY','CC-BY-SA']}
Theme:
  attributes:
    title: {type:string}
    css_definition: {type:text}
```

Figure 2: Instantiation of the WikiApp DSL representing the SlideWiki model.

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.

The database schema generated by Wikifier for SlideWiki example is depicted in Figure 4. 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. SLIDEWIKI: AN EXAMPLE WIKIAPP IMPLEMENTATION

In this section we describe with SlideWiki a concrete WikiApp implementation, which we created to demonstrate the effectiveness and efficiency of the WikiApp approach[5]. The main idea of SlideWiki is to enable the crowd-sourcing of educational content, in particular presentations. The SlideWiki data model was already introduced in Example 1 and shows the relatively complex relationships between decks, slides, media assets and themes. The rationale behind creating the domain-specific SlideWiki application is to hide this complexity as much as possible from its users.

[5]SlideWiki is available publicly at `http://slidewiki.aksw.org` with a accompanying video screencast at: `http://slidewiki.aksw.org/video/slidewiki.mp4`

Our SlideWiki application makes extensive use of the model-view-controller (MVC) architecture pattern. The MVC architecture enables the decoupling of the user interface, program logic and database controllers and thus allows developers to maintain each of these components separately. The overall architecture of SlideWiki is presented in Figure 4. The view layer comprises various types of user interactions such as viewing & playing presentations, annotating content objects (i.e. adding tags and comments to slides and presentations), access control (i.e. creating accounts in order to author content). Since SlideWiki is implemented as a typical Web application the view layer is accessed by the user via a Web browser using the HTTP protocol, AJAX and HTML, CSS documents. A typical URL for accessing a particular slide in the deck is depicted in Figure 3. Using this URL, the user accesses the *Viewing & Playing* component of the view layer. The request is dispatched to a particular controller, which in turn employs SlideWiki backend classes and methods to interact with the persistent storage backend (a relational database in this case).

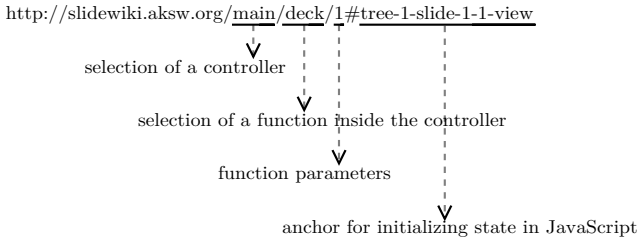

Figure 3: Mapping of URLs to MVC actions in SlideWiki.

In both the front-end and back-end part of SlideWiki we make extensive use of existing open-source components. On the back-end side, we use components for zip archive handling (required for PPTX and HTML import and export functions), CAPTCHA generation (for restricting machine access), and image manipulation components (e.g. for the creation of slide previews). The SlideWiki front-end (as shown in the screenshots in Figure 6) relies upon *deck.js*[6] for rendering and playing of presentations, *TinyMCE*[7] as slide authoring widget, *MathJax* for rendering embedded *LaTeX* formulas[8], as well as the foundational *jQuery*[9] framework together with various extensions[10]. Also, the SlideWiki front-end uses *Twitter bootstrap*[11] for CSS styling and design. In the sequel we describe some important aspects in more detail.

4.1 Client-side MVC

In addition to overall MVC pattern, SlideWiki utilizes a client-side MVC approach (implemented in JavaScript and running inside the users Web browser) to provide users with a rich and interactive user interface. As described in Figure 3, there is a hash fragment in the request URL which acts as an input for the client-side MVC handler. This fragment consists of an *identifier* and an *action* name. The identifier itself has four parts which are combined based on the following pattern: `tree-{container_deck_id}-{content_type}-{content_id}-{content_position}`. For example, `tree-1-slide-5-2-view` refers to the `view` action which is assigned to the slide with id 5, located at second position of deck with id 1.

[6]http://imakewebthings.github.com/deck.js/
[7]http://tinymce.com/
[8]http://www.mathjax.org/
[9]http://jquery.com/
[10]FancyBox, jQuery Templates, jQuery UI, jQuery jsTree
[11]http://twitter.github.com/bootstrap/

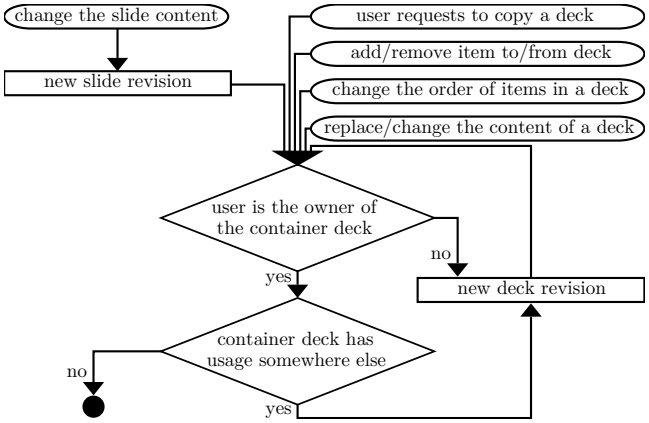

Figure 5: Decision flow during the creation of new slide and deck revisions.

The client-side MVC handler as (singleton) controller listens to the hash fragment and once a change has occurred the handler triggers the corresponding actions. Each action has a JavaScript template (implemented using *jQuery templates*) with the corresponding variable place holders. For each action an Ajax call is made and the results are returned to the controller in JSON format. Subsequently, the controller fills the templates with the results and renders them in the browser.

4.2 Revision control

Although substantially simplified by the WikiApp data model revision control is an important issue in SlideWiki. SlideWiki generally follows the approach, that content objects are not updated or deleted, but instead, in accordance with the WikiApp data model, new revisions will be created. As shown in Figure 5, there are different cases in SlideWiki for which new slide or deck 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.

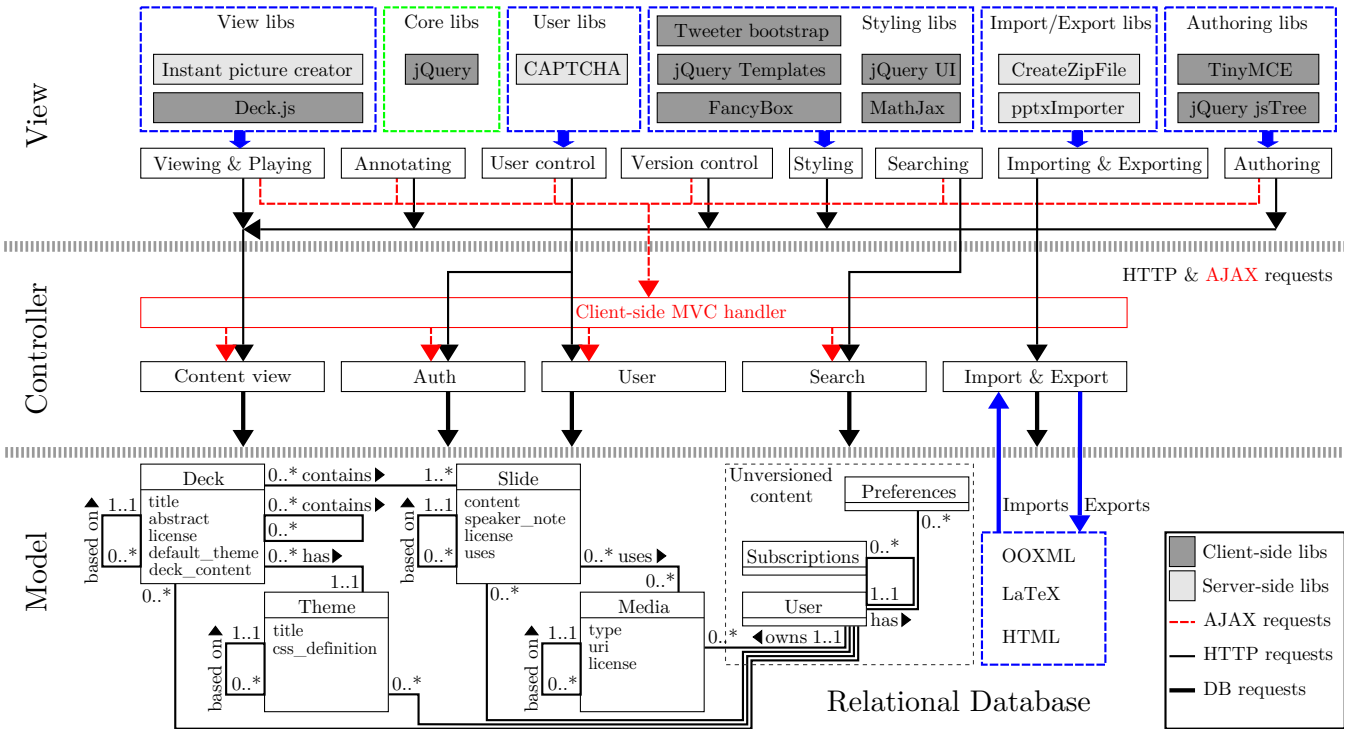

Figure 4: Bird's eye view on the SlideWiki MVC architecture.

- 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.

4.3 Linked Data Interface

WikiApp implementations can be easily equipped with a Linked Data interface. We employed the RDB2RDF mapping tool Triplify [1] 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`

4.4 Search

There are three ways of searching in SlideWiki: by keywords, by metadata and by user (who contributed or follows certain content). We combined keywords and tags search so that users can either 1. search by keywords and then add a tag filter, or 2. show all slides or decks having the tag and then running an additional keyword search on the results. In both cases an ordering a user might have applied is preserved for subsequent searches. It is possible to show the last or the most popular revisions in search results both for slides and decks. The keyword search panel is always visible on the top of each page (cf. Figure 6). Search results are divided into two tabs: decks and slides, each with its own pager. Results can be ordered by title, creation date, user or popularity. The search is based on the MySQL text indexing feature with the `MATCH...AGAINST` function, executed in boolean mode. For implementing the full-text search, the `title`, `slide content` and `deck abstract` database columns are full-text indexed. For SlideWiki we decrease the minimal length, that a word must have to be indexed to 3 characters in order to support search for acronyms such as OWL or RDF. Also there is a list of the presentations and slides contributed or followed by the certain user in her profile page, that helps users to quickly reach decks or slides they work on.

4.5 Popularity

To obtain the popularity of presentations and slides we use weighted values of three indicators: *number of subscribers* relatively to the total number of users, *usage* in other decks relatively to the total number of decks and *number of views* relatively to the total number of visitors. We chose the number of subscribers, excluding its owner, as the most important indicator and set its weight to 0.5. The usage in other decks is also an important indicator (similar to Google's page rank), but we decided to exclude the usage count in decks with the same owner. We weighted this indicator with a

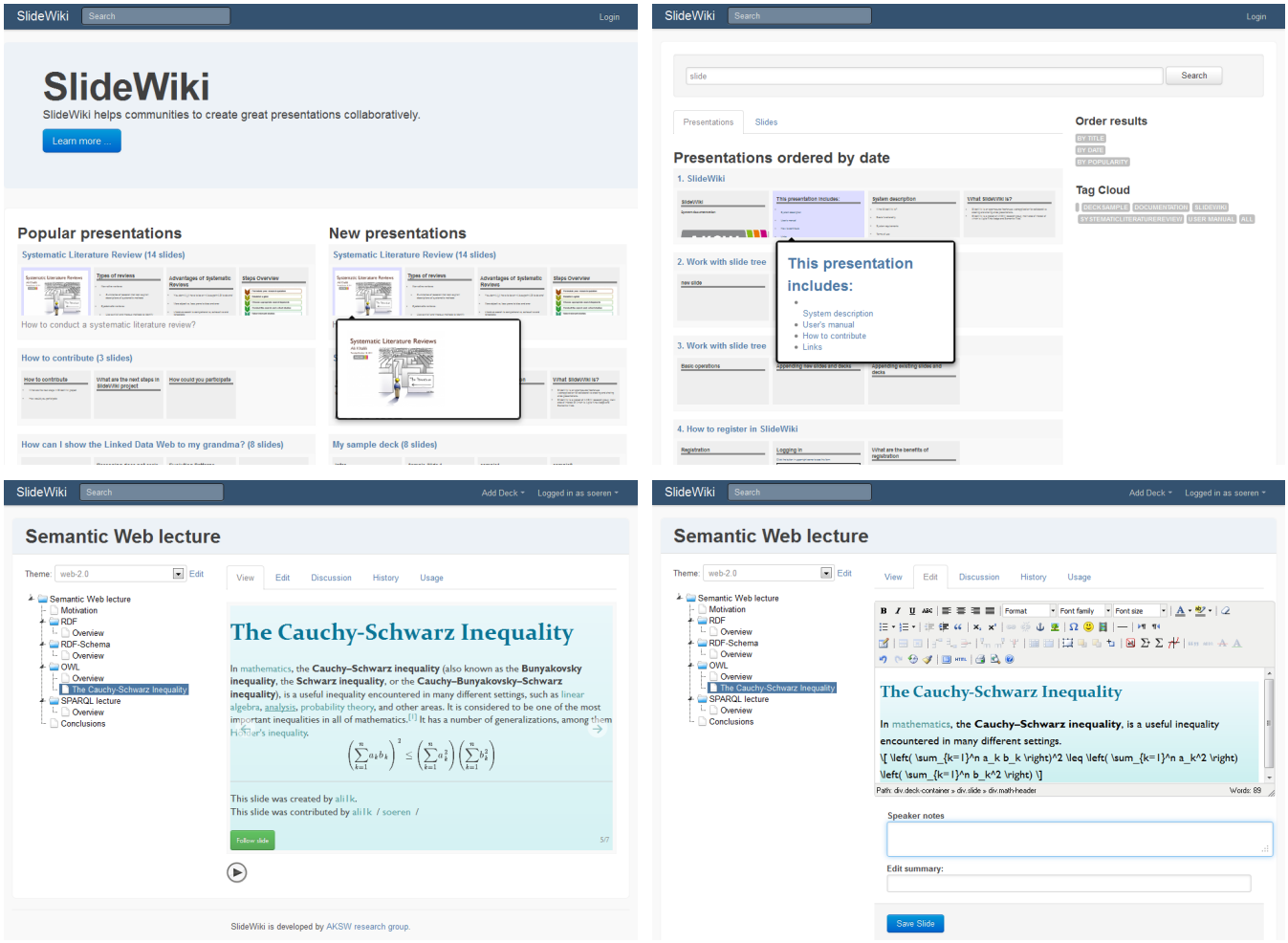

Figure 6: Four screenshots of the SlideWiki application: (Top-left) The SlideWiki homepage showing new and popular presentations. (Top-right) Full-text search and faceted browsing of search results. (Bottom-left) Presentation browsing: decks can be structured as trees with sub-decks and slides. (Bottom-right) Slide authoring: LaTeX code can be directly embedded; speaker notes and revision comments can be added.

factor of 0.3, since this factor could be compromised by the creation of secondary accounts and using these items in new decks. The number of views has a relatively low weight (0.2), owner and robot views are excluded. The final formula is represented is then: $P = 0.5 * \frac{S}{U} + 0.3 * \frac{R}{D} + 0.2 * \frac{V}{I}$, with P – popularity of the deck or the slide, S – number of subscribers of the deck or the slide excluding its owner, U – total number of users, R – number of usage in other decks excluding decks with the same owner, D – total number of decks, V - number of views excluding robots and the owner, I – total number of visitors, excluding robots and the owner.

4.6 Import and Export

A crucial feature of SlideWiki is an ability to import and export data into different formats. Without the possibility to import and export data (i.e. making backups and transferring data to other data mediums or applications) user will be discouraged from contributing and maintaining data on the platform. The main data format used in SlideWiki is HTML. However, there are other popular presentation formats commonly used by desktop application users, such as Office Open XML (ECMA-376 or ISO/IEC 29500) and LaTeX. Thus we implemented importing and exporting functionality for above-mentioned formats. Currently, SlideWiki supports importing from Microsoft Office implementation of ECMA-376 format (i.e. files with .pptx extension) and exporting to the deck.js HTML+JS format. LaTeX and OpenOffice ECMA-376 implementation support are prepared but not yet completed. The import/export interface utilizes the MVC architecture of WikiApp and has itself a modular structure. Each module corresponds to a particular format (e.g. Office Open XML, LaTeX, HTML), thus facilitating the reuse existing import/export libraries. Modules are linked to SlideWiki through *Import & Export* component of the controller.

5. EVALUATION

We evaluate the WikiApp concept according to two dimensions: Firstly, we verify our claim, that WikiApp implementations will scale better than comparable semantic wiki applications with a synthetic benchmark of the SlideWiki WikiApp implementation. Secondly, we show with a user

Table 1: Benchmarking results (all timings in ms).

Operation	SlideWiki			Ontowiki		
Data (decks)	**100**	**500**	**1000**	**100**	**500**	**1000**
(slides)	2,905	16,459	32,629	2,905	16,459	32,629
Import (average per slide)						
processing	771	934	951	-	-	-
database	1	2	3	-	-	-
total	772	936	954	-	-	-
Export (average per slide)						
total	54	156	209	-	-	-
Search for decks (average)						
database	6	8	10	5	7	12
processing	36	37	45	36	37	45
total	42	45	55	41	44	57
Search for slides (average)						
database	7	17	49	59	329	642
processing	118	238	376	118	238	376
total	125	255	425	177	667	1108
Move (average)						
deck	4	4	5	1	2	3
slide	3	4	5	3	4	5
Show deck content (average per slide)						
deck	10	35	59	12	40	81
Create new revision (average)						
deck	14	17	21	2	3	4
slide	1	1	2	3	5	7

study that by creating domain-specific applications the complexity of WikiApp implementations based on comprehensive semi-structured data can be effectively hidden from end-users.

Synthetic benchmarking. To measure the system performance, 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. The results of the benchmarking are summarized in the Table 1. 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 4.3), into OntoWiki [9] (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 7. Based on the measurement results, we can conclude, that SlideWiki performance 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*.

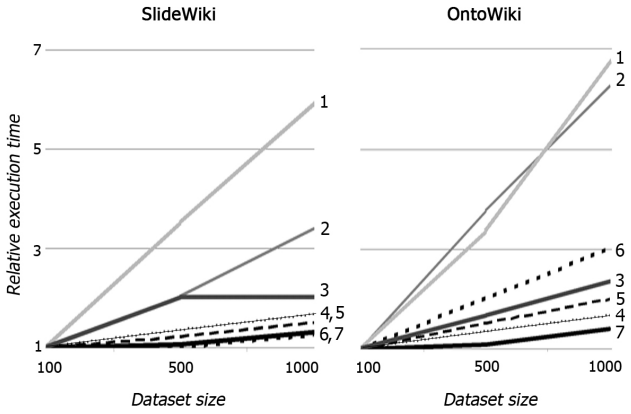

Figure 7: Benchmarking results: Execution time relatively to 100-deck dataset. Legend: 1-*show deck*, 2-*search slides*, 3-*new slide revision*, 4-*move slide*, 5-*new deck revision*, 6-*move deck*, 7-*search deck*.

Usability Evaluation. In order to determine whether we succeeded to effectively hide SlideWiki's data model complexity, we performed a usability user study with 13 subjects. Subjects were drawn from the members of AKSW research group, the computer science department at the university of Leipzig. We first showed them a tutorial video of using different features of SlideWiki then asked each one to create a presentation with SlideWiki. After finishing the task, we asked the participants to fill out a questionnaire which consisted of three parts: demographic questions, feature usage questions and usability experience questions. Table 2 summarizes the different SlideWiki features as well as their usage during the evaluation. We used the *System Usability Scale* (SUS) [14] to grade the usability of SlideWiki. SUS is a standardized, simple, ten-item Likert scale-based questionnaire[12] giving a global view of subjective assessments of usability. It yields a single number in the range of 0 to 100 which represents a composite measure of the overall usability of the system. The results of our survey showed a mean usability score of 69.62 for SlideWiki which indicates a reasonable level of usability. Of course, this is a very simplified view on usability and we expect even better results could be achieved by putting more effort into the SlideWiki development (the development of SlideWiki only consumed 5 person months). However, our goal was to demonstrate that WikiApp implementations with good usability characteristics can be created with relatively limited effort. In addition to quantitative results, we also collected a number of user suggestions. For instance some users suggested improving the WYSIWYG editor for adding shapes, providing autosave feature, supporting more import/export formats, defining user groups etc.

6. RELATED WORK

Related work can be roughly divided into wiki-based knowledge engineering, semantic wikis, model-driven web engineering and collaborative E-Learning content creation.

[12] `www.usabilitynet.org/trump/documents/Suschapt.doc`

Feature	Usage
WYSIWYG slide authoring	76.92%
Importing pptx presentations	46.15%
Using LaTeX in slide content	58.85%
Using/Creating themes for decks	46.15%
Searching slides/decks	76.92%
Adding existing slides/decks to your deck	69.23%
Following slides/decks/users	53.85%
Contributing to other's slides/decks	53.85%
Using other revisions of slides/decks	76.92%
Playing slides/decks	**92.31**%
Downloading the decks for offline preview	**38.46**%
Adding comments about slides/decks	61.54%

Table 2: SlideWiki feature usage per user.

Wiki-based Collaborative Knowledge Engineering. The importance of wikis for collaborative knowledge engineering is meanwhile widely acknowledged. In [19], 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, [8] 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 [15] 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 applications. The wiki paradigm was meanwhile also applied to domain-specific applications, such as, for example, *Adhocracy*[13] – a policy drafting tool for distributed groups. In a way WikiApp generalizes such applications and provides a common conceptual and methodological framework for their development.

Semantic Wikis. Another approach to combine wiki technology with structured representation are semantic wikis [22]. There are two types of semantic wikis. Semantic text wikis, such as Semantic MediaWiki [11] or KiWi [23] are based on semantic annotations of the textual content. Semantic data wikis, such as OntoWiki [9], 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 [3]. 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 Erfurt API, RDFauthor and Semantic Pingback, which evolved OntoWiki into a framework for Web Application development [9].

[13] `http://trac.adhocracy.cc/`

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 [6] or more recently WebForm Diagram [24] 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 [17]. WebML 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 [10]. There are a number of recent advances with regard to integrating specific aspects into model-driven Web application engineering, such as evolution and adding of concerns [16]; requirements and user-interface-driven development [20] or usability [5]. We deem WikiApp due to its focus on the data structures being orthogonal to most of these approaches and thus being easily combinable.

Collaborative creation of E-Learning content. Our SlideWiki WikiApp implementation can be viewed as a tool for the large-scale collaborative creation of educational content. The importance of creating reusable and re-purposable E-learning objects is meanwhile widely accepted by the E-Learning community [4]. However, it seems that most of the work addresses the learning object reuse problem rather by means of semantic meta-data annotations, content tagging and packaging rather than by creating richly structured, reusable learning objects from the ground. The importance of creating learning objects already with reuse in mind was, for example, stated by [18]: 'Content ... should be represented not as an object of study but rather as necessary elements towards a series of objectives that will be discovered in the course of various tests.'. There are only few approaches for the direct authoring of reusable content, such as, for example, learning examples creation [12] or semantic structuring and annotation of video fragments [2]. With SlideWiki we aimed to create an educational content creation platform grounded on reusable content authoring and large-scale community collaboration (or crowd-sourcing).

7. CONCLUSIONS

In this paper we presented the WikiApp approach and accompanying tools for creating domain-specific wiki-like applications. WikiApp addresses weaknesses of conventional text-oriented as well as semantic wikis. Compared to 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 database technology). In a certain way the WikiApp approach can also be viewed as a complex *design pattern* for the creation of crowd-sourcing enabled Web applications.

We see this effort as the first step in a larger research and engineering agenda: firstly, we expect the WikiApp concept to be applied to many more application domains beyond our current application for educational content with SlideWiki. In particular, we aim to extend SlideWiki to educational content beyond decks and slides, such as assessment tests. Secondly, we want to increase the support for the model-driven development of WikiApp implementations. In particular, we plan extensions to the WikiApp DSL and the Wikifier transformation, which will enable the automatic creation of user interface components and the use of cloud computing infrastructure.

8. REFERENCES

[1] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: Light-weight linked data publication from relational databases. In *18th Int. Conf. on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 621–630. ACM, 2009.

[2] Elena García Barriocanal, Miguel-Ángel Sicilia, Salvador Sánchez Alonso, and Miltiadis D. Lytras. Semantic annotation of video fragments as learning objects. *Interactive Learning Environments*, 19(1):25–44, 2011.

[3] Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[4] Vladan Devedzic. *Semantic Web and Education (Integrated Series in Information Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] Adrian Fernandez, Emilio Insfrán, and Silvia Mara Abrahão. Towards a usability evaluation process for model-driven web development. In *I-USED*, 2009.

[6] Piero Fraternali and Paolo Paolini. Model-driven development of web applications: the autoweb system. *ACM Trans. Inf. Syst.*, 18:323–382, October 2000.

[7] Martin Gaedke, Michael Grossniklaus, and Oscar Díaz, editors. *Web Engineering, 9th International Conference, ICWE 2009*, volume 5648 of *LNCS*. Springer, 2009.

[8] Anja Haake, Stephan Lukosch, and Till Schümmer. Wiki-templates: adding structure support to wikis on demand. In Dirk Riehle, editor, *Int. Sym. Wikis*, pages 41–51. ACM, 2005.

[9] Norman Heino, Sebastian Dietzold, Michael Martin, and Sören Auer. Developing semantic web applications with the ontowiki framework. In *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 61–77. Springer, Berlin / Heidelberg, 2009.

[10] Christian Kroiss, Nora Koch, and Alexander Knapp. Uwe4jsf: A model-driven generation approach for web applications. In Gaedke et al. [7], pages 493–496.

[11] Markus Krötzsch, Denny Vrandečić, Max Völkel, Heiko Haller, and Rudi Studer. Semantic Wikipedia. *Journal of Web Semantics*, 5(4):251–261, 2007.

[12] Yen-Hung Kuo, Qing Tan, Kinshuk, Yueh-Min Huang, Tzu-Chien Liu, and Maiga Chang. Collaborative creation of authentic examples with location for u-learning. In *e-Learning*, pages 16–20, 2008.

[13] Bo Leuf and Ward Cunningham. *The Wiki way: quick collaboration on the Web*. Addison-Wesley, London, 2001.

[14] James Lewis and Jeff Sauro. The Factor Structure of the System Usability Scale. In *Human Centered Design*, volume 5619 of *LNCS*, chapter 12, pages 94–103. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009.

[15] Florian Matthes, Christian Neubert, and Alexander Steinhoff. Hybrid wikis: Empowering users to collaboratively structure information. In *ICSOFT (1)*, pages 250–259. SciTePress, 2011.

[16] Nathalie Moreno, Santiago Meliá, Nora Koch, and Antonio Vallecillo. Addressing new concerns in model-driven web engineering approaches. In *WISE*, volume 5175 of *LNCS*, pages 426–442. Springer, 2008.

[17] Richard F. Paige and Bertrand Meyer, editors. *Objects, Components, Models and Patterns, 46th International Conference, TOOLS EUROPE 2008*, volume 11 of *LNBI*. Springer, 2008.

[18] Nieves Pedreira, José Ramón Méndez Salgueiro, and Manuel Martínez Carballo. E-learning in new technologies. In *Encyclopedia of Artificial Intelligence*, pages 532–535. IGI Global, 2009.

[19] Debbie Richards. A social software/web 2.0 approach to collaborative knowledge engineering. *Inf. Sci.*, 179(15):2515–2523, 2009.

[20] José Matías Rivero, Julián Grigera, Gustavo Rossi, Esteban Robles Luna, and Nora Koch. Improving agility in model-driven web engineering. In *CAiSE Forum*, volume 734 of *CEUR Workshop Proceedings*, pages 163–170. CEUR-WS.org, 2011.

[21] Satya S. Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, and Ahmed Ezzat. A survey of current approaches for mapping of relational databases to rdf, 01 2009.

[22] Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel. Semantic wikis. *IEEE Software*, 25(4):8–11, 2008.

[23] Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint, and Stephanie Stroka. Kiwi - a platform for semantic social software. In *SemWiki*, volume 464 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[24] Jurriaan Souer, Thijs Kupers, Remko Helms, and Sjaak Brinkkemper. Model-driven web engineering for the automated configuration of web content management systems. In Gaedke et al. [7], pages 121–135.