Ideas and Improvements for Semantic Wikis

Jochen Fischer, Zeno Gantner, Steffen Rendle,
Manuel Stritt, and Lars Schmidt-Thieme

Department of Computer Science, University of Freiburg
Georges-Köhler-Allee 51, D-79110 Freiburg, Germany
{jocfisch, ganter ,stritt, lst}@informatik.uni-freiburg.de,
steffen@rendle.de

Abstract. We present an architecture for combining wikis containing hypertext with ontologies containing formal, structured information. A web-based ontology editor that supports collaborative work through versioning, transactions and management of simultaneous modifications is used for ontology evolution. In wiki pages, ontology information can be used to render dynamic content and answer user queries. Furthermore, query templates are introduced that simplify the use of queries for inexperienced users. The architecture allows easy integration with existing ontology frameworks and wiki engines. The usefulness of the approach is demonstrated by a prototypical implementation as well as a small case study.

1 Introduction

A wiki, short for WikiWikiWeb [1], is a website that allows collaborative creation and editing of hypertext content, usually expressed in a simple markup language. *COW*, *Combining Ontologies with Wikis*[1], is a novel approach to build a *semantic wiki*, by bringing together two different concepts: easy content evolution with the help of wikis, and formal knowledge representation using ontologies. We use the KAON tool suite [2] as back-end for an ontology editor and a query processor; a simple text wiki engine complements the system's functionality. Our approach is different from other semantic wikis in two aspects: The ontology data is edited and stored outside the text wiki, and we implemented so-called query templates, which can be particularly useful for inexperienced users (figure 1).

Although there are already some approaches to using wiki-like systems (see section 2) in the context of ontologies [3] and the Semantic Web [4], still a lot remains to be explored in this field. Requiring detailed knowledge about languages like RDF or OWL would contradict an important aspect of the wiki idea: simplicity and ease of use. It would be comparable to forcing the use of full HTML in text wikis, instead of a more user-friendly and minimalistic syntax.

[1] http://www.informatik.uni-freiburg.de/cgnm/software/cow/, available under the terms of the GNU General Public License.

Y. Sure and J. Domingue (Eds.): ESWC 2006, LNCS 4011, pp. 650–663, 2006.

This paper is structured as follows. In the beginning, we give an overview on semantic wikis developed so far. Then we present COW's architecture, and how ontology editing works in the application, with special focus on problems that may occur during simultaneous editing. Next, we introduce COW's query functionality. Finally, we show how these features could be used in a concrete application, a small biographical lexicon.

2 Related Work

Platypus Wiki [5] is a wiki engine that allows entering RDF and OWL statements in addition to natural language text. Unlike our system, it treats the statements in the ontology language as text, instead of providing information on a conceptual level. It also lacks ontology querying, which COW offers both interactively and integrated in the text wiki.

Rhizome[2] [6] is a wiki-like content management system built on top of the RDF application server Raccoon. Metadata is described by the RxML language. RxPath is used for querying the RDF model. Again, the drawbacks are the rather complicated RxML, the difficulty of querying, and the lack of support for collaborative work.

MediaWiki[3], the software behind Wikipedia, enables the user to categorize articles, or, more generally, wiki pages. Categories themselves can also be included in other categories. While this is useful for grouping articles, the associations encoded directly and indirectly using the category feature are not exploited e.g. for improving search results. Beyond categorization and links, there is no possibility of adding formal, well-defined information to the content. COW allows building ontologies that are more complex than a taxonomy of categories. There is an approach to extend MediaWiki with so-called "semantic links", which encode relations between page subjects [7].

WikSAR [8] and SemWiki[4] are wiki engines which allow entering semantic content in the normal text. Both systems offer query mechanisms, however they lack query templates. Again semantic information has to be represented in a formal language and is stored as part of normal wiki pages.

Similar to other wikis COW adopts the web-based user interface from semantic wikis like Platypus or Rhizome. Hence the system can be accessed by any web browser. In contrast to other semantic wikis like WikSAR, a frame-based view for ontology editing is implemented, similarly to Protégé [9] or WebODE [10]. The ontology data itself is not stored in a text wiki, but in a separate database, where it is kept consistent. Several existing text wikis can be adapted to work with the software. To make use of the ontology data, COW uses the KAON query language and extends it by query templates, which will be described later in this article.

[2] http://rhizome.liminalzone.org/
[3] http://www.mediawiki.org/
[4] http://www.dello.net/semwiki/

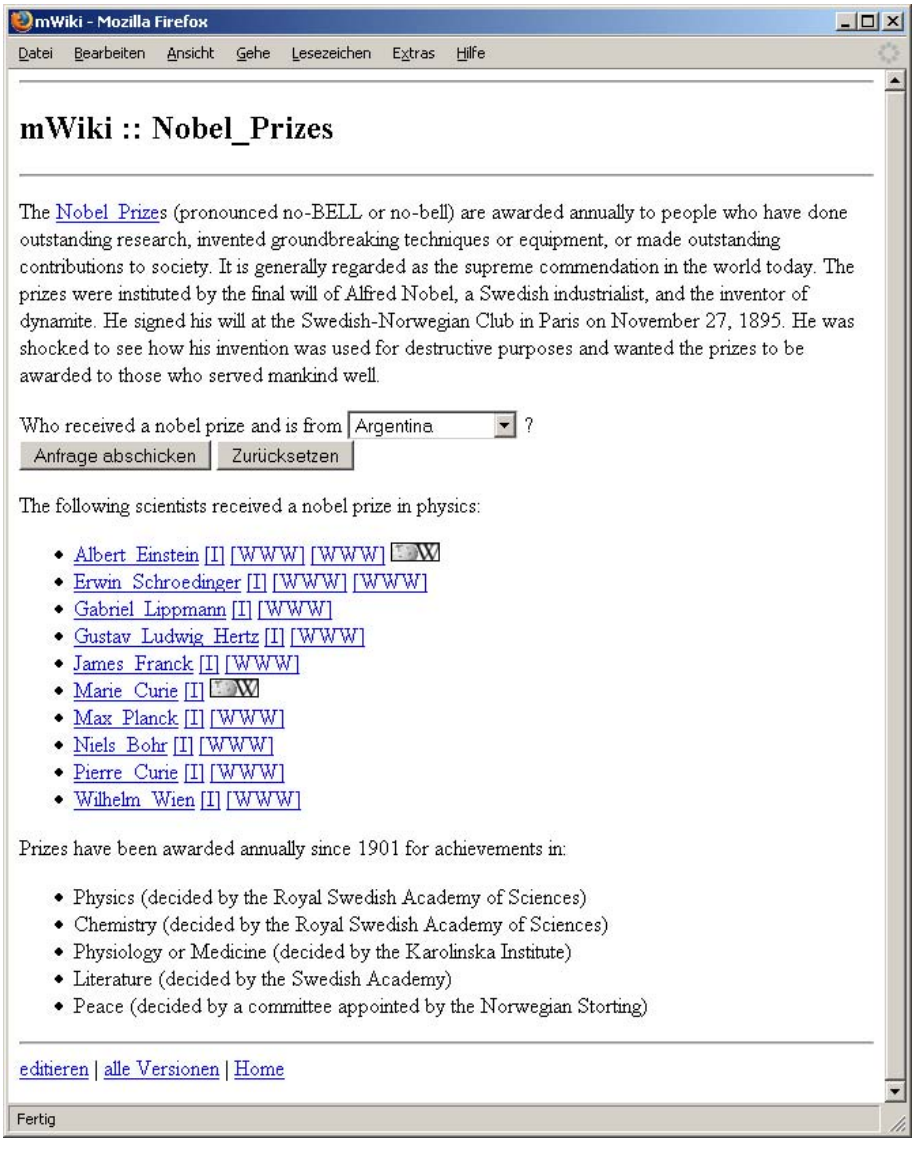

Fig. 1. COW renders normal text (top and bottom of the page), query templates (the question with the pull-down menu), and dynamic content (list in the center)

3 Architecture

COW has a multi-tier architecture (fig. 2) consisting of the following parts: persistence layer, ontology layer, abstraction layer, and an HTML-based user interface.

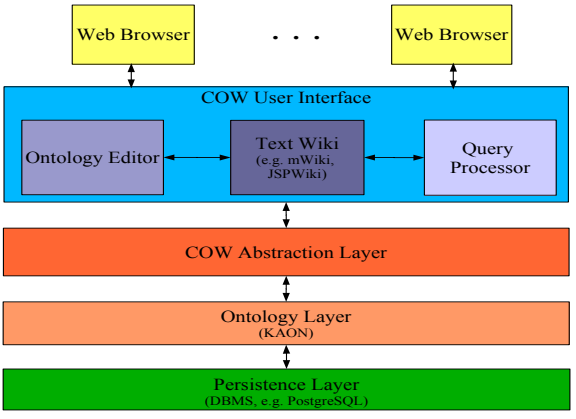

Fig. 2. COW Architecture

All end-user interaction with COW is done with a standard web browser using only plain HTML. Neither Java applets nor JavaScript are used. Thus we support one of the wiki principles, namely low client-side requirements.

Abstraction and Ontology Layer

An abstraction layer for language refinement and version management separates the wiki and ontology editor components from a general purpose ontology framework. This layer provides a lightweight, string-based interface to communicate with the front-end, stores the ontology permanently in a database, and is able to handle queries.

COW uses the ontology framework KAON [2] to implement the interface of the abstraction layer. A useful feature of KAON is the evolution log, a list stored in a meta-ontology where all ontology changes are logged. This log enables us to restore any previous version of the ontology. The framework does not allow inconsistent states of the ontology. After applying a change, either the ontology is consistent or an exception will be thrown (and caught). In combination with the locking mechanism, which will be described later, this ensures effective collaborative editing of the ontology. Internally COW uses the ontology language RDF plus extensions like transitive and inverse properties. To support the reusing ontology information we provide a dynamic page which exports the complete ontology in OWL.

Integration into Existing Wiki Engines

Although we implemented a minimalistic wiki engine, we designed the other components of the system to be as independent as possible from the used wiki engine. Little work is necessary to combine the ontology editor and the query engine with the text wiki.

To demonstrate the independence of the system, we also integrated COW's functionality into JSPWiki[5]. A small plugin encapsulating the query functionality was developed. The linking is realized by taking advantage of the concept of interwiki links[6]: E.g., the code `[Concept:Person]` is rendered to an HTML link pointing to the ontology view of the concept `Person`.

4 Ontology Editing and Browsing

In a classical wiki scenario one might identify two groups of users: readers and contributors. The first ones use information already stored in the system. The later ones contribute information to the system. Wikipedia reports that at least 99% of its daily visitors are readers and only less than 1% are contributors[7].

For semantic wikis, a third group can be detected which we call "experts": People with sufficient understanding of knowledge representation mechanisms to contribute to the semantic data of the web site, especially the ontology structure. Note that these three groups are not disjoint: Both text contributors and experts usually also act as readers, although they should not be regarded as typical members of the group.

For the success of a semantic wiki it is crucial that the system is capable of serving the needs of all three groups. Because the group of readers is the vast majority the overall success of a system depends mostly on them. That is why the systems should be designed to be easy and intuitive for these readers. Secondly it is important that inexperienced contributors can work on the system, so that the content can grow fast.

We think that today's semantic wikis address mostly the group of experts. By separating the ontology storage from the wiki text, we avoid confusion among the contributors who are not experts.

In order to allow inexperienced users to help populating the ontology, COW has a slot-based graphical ontology editor. This way instances may be created without much expertise. It is important to note that no user has to learn a formal language like RDF or OWL.

We created a simple web-based ontology editor component, because, at the time of COW's initial development, there was no such system available. Today, we might consider using pOWL [11].

5 Ontology Locking, Transactions, and Versioning

For collaborative work on ontologies, our system supports ontology versioning, transactions and locking of editing sessions. The objective of these features is to

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

[6] See http://c2.com/cgi/wiki?InterWiki for an explanation.

[7] Estimation based on last available visitor statistics of October 2004. There are 917,000 daily visitors and there are only about 11,000 contributors that have ever edited more than 5 pages. http://en.wikipedia.org/wikistats/DE/-TablesWikipediaZZ.htm

Fig. 3. Screenshot of the editor with the ontology browser as active module

keep the ontology consistent and to provide an natural and simple workflow to the user.

Locking Strategy for Editing Ontology Elements

A central problem of simultaneous editing of ontologies is how to synchronize different editing sessions. In text wikis, every page is the atomic element for editing operations. When a user starts to edit a page, the wiki system locks this page for other editing operations until the user applies his changes or after a certain time threshold expires. Other wiki engines use a "first come, first served" strategy for check-ins. Locking mechanisms for ontology systems are more complicated because of dependencies between several entities. For example if user A edits instance `I` of concept `C` and user B adds a slot with domain `C`, the editing session of user B depends on A's changes. Therefore, locking of single entities is not sufficient. On the other hand, locking the ontology as a whole obviously is an obstacle for concurrent editing, especially for large ontologies that many people want to work on simultaneously. Furthermore, as locking should be applied only if necessary, and in wikis users sometimes start the edit mode, but do not apply any changes, any locking mechanism which is applied at check-out time is unnecessary restrictive.

COW's checks are performed when the user commits changes. These checks guarantee that the result of the check-in is comprehensible for the user. The changes to an entity are refused by the system if the editor view of this entity has changed in the meantime. With this strategy users always know the exact effects of their editing operations. All dependencies causing a change of the edit

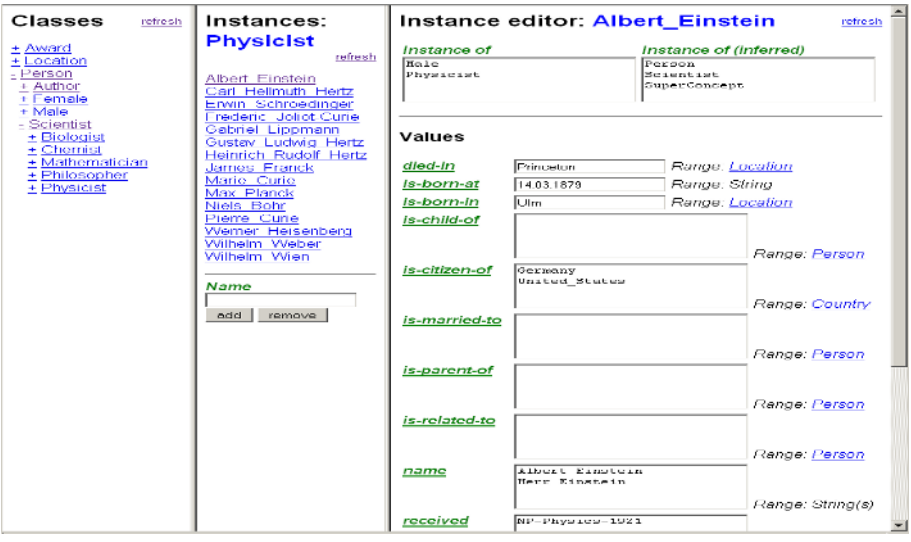

Fig. 4. Screenshot of the instance editor with instance `Albert Einstein` in edit mode

view are considered, even inferred ones. Fig. 5 shows how our locking mechanism works.

We have tested our strategy in our case study, where we have built and populated the ontology simultaneously. The system had to refuse editing operations quite rarely even though our ontology changed often. Generally, our locking mechanism stayed in the background and did not disturb working collaboratively.

Transactions for Ontology Modification

In RDF-based knowledge representation, the smallest unit of information is an RDF triple. As triples of an editing session depend on each other, changes on the knowledge base should be performed according to the ACID principles[8]. Thus we offer a transaction mode for ontologies, which guarantees that either all changes are applied or none.

Ontology Versioning

The most important design issue with respect to ontology versioning is the granularity of the versioning method. Versioning every entity – concepts, instances, properties – on its own would imply that previous versions of each entity could be restored independently from the rest of the ontology. Unfortunately, this would be very complicated because of dependencies between ontology entities. Even if it were possible to find a strategy that can manage different versions of entities

[8] Atomicity, consistency, isolation, and durability are the key properties of transactions in database management systems. See [12].

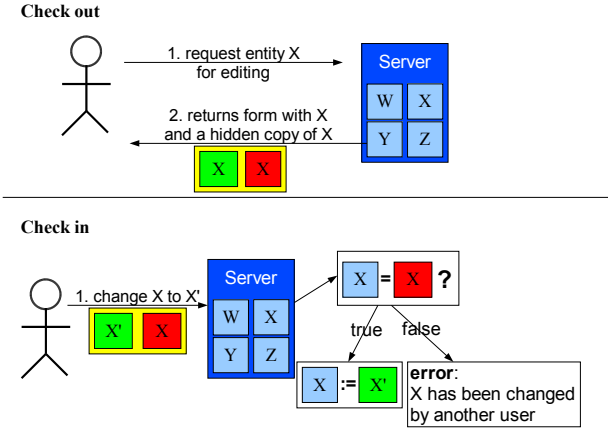

Fig. 5. Demonstration of the locking mechanism. Saving X is accepted if and only if the editing view, including possible inferred values, of X has not been changed.

and handles the dependencies in a way that ensures a consistent ontology [13], it would be everything but comprehensible to the user.

Our approach versions the whole ontology. If users want to restore old versions, they have to set the complete ontology to the old state. With regard to simplicity, we think that this is an effective way that is traceable by the users. Of course a roll-back to an older version of the complete ontology might imply a lot of changes. It's out of the focus of this paper to implement state-of-the-art ontology-versioning, which is still an active topic of research. For example, a more elabotative approach could version independent domains in an ontology seperately, which might be an interesting feature for future work.

6 Querying Ontologies Inside the Wiki

An application for browsing and editing both a text collection and an ontology requires the possibility to search the text as well as query the content of the ontology. In COW, the users specify queries, either using a dedicated page, or by directly embedding them into wiki pages. The query results, usually instances, provide both hyperlinks to the article pages and to the ontology browser.

Normal queries are statements in the KAON query language, which will be described briefly below. Additionally, we developed *query templates* that can be added to wiki pages. These templates are queries with free variable parameters, which have to be filled by the user executing the query. Note that this is realized by passing queries to the underlying ontology framework. Other query languages could be supported by using another ontology framework or by adding components supporting the language.

KAON offers a comfortable language for querying ontologies. In analogy to SQL, which is a closed language over relations, the language is closed over conceptual descriptions.

The simplest queries are questions about facts that are directly stored in an ontology, like the property values of a given instance. If a user wants to know when Albert Einstein was born, he has just to enter

```
<#day-of-birth> IN:1 !#Albert_Einstein!
```

Besides this, the system can also answer queries by using its inference mechanism:

```
[#Physicist] AND
    SOME(<#is-born-in>.<#located-in>=!#Europe!)
```

`#Albert_Einstein` will be in the result set, even though it is not directly stored in the ontology that he was born in Europe. It is not even stored that his home town, Ulm, is located in Europe, but KAON's reasoning engine derived this from the information that Ulm is located in Germany, Germany is located in Europe, and the `<#located-in>` property is transitive. As the `<#located-in>` property is not reflexive, the query will not yield persons for whom it is only stated that they were born in Europe. Thus to get correct results, one has to enhance the query a little bit further:

```
[#Physicist] AND (SOME(<#is-born-in>.<#located-in>=!#Europe!)
                  OR SOME(<#is-born-in>=!#Europe!))
```

While KAON's query language is fairly simple and offers lots of useful constructs, it still has disadvantages for its use in a system that wants to be user-friendly for non-experts which it will never overcome: It is a formal language with strict syntax and semantics, and there might be subtle differences between the user's conceptual model and the actual ontology which will have a undesirable or at least unexpected impact on the query results.

Because we do not see a viable alternative to a formal query language, we decided to use such a language in COW, to empower at least the users having the necessary expert knowledge to use the query feature.

As the queries are stored on the wiki pages, other users can look them up and use them as examples or boilerplates for constructing their own queries. By and by, a collection of useful queries might be accumulated in the wiki.

Query Templates

One way to facilitate the reuse of queries is *query templates*, queries with placeholders waiting to be filled by the user with instances or literal values. Expert users may create typical questions on important concepts of the ontology, e.g. persons and awards in our case study, which then can be used by all users who want to get information from the system. As a wiki also serves as a communication platform for its user community, people also might approach others for getting certain queries implemented. This can make a semantic wiki a more useful tool than a normal collaborative editor for ontologies.

A *query template* is a triple (q, t, s), where

- q is a query containing placeholder variables enclosed in dollar signs ("$"), each variable may occur several times in the query;
- t is a string containing a natural language phrase - usually a question - representing the query, each variable in q occurs exactly once in t together with a type statement;
- s is a set of tuples containing the mappings from all non-string variables to the queries that yield the set of (property) instances that will be presented to the user as values to be selected for the variables.

A placeholder variable has the form `$NAME$`, a variable with a type statement has the form `$NAME:type$`, where `type` can be `instance`, `concept`, `property`, or `literal`. For example:

```
q='SOME (<#worked-with>=$PERSON$)'
t='Who worked with $PERSON:instance$'
s={('PERSON','[#Person]')}
```

In the wiki, this query template will be displayed as the question formulated above. All the variables are replaced by pull-down menus or input fields inside the question. After filling out or selecting all the items, the user will be redirected to a page where the results of the query are presented. We think that other semantic wikis can profit from the concept of query templates as well, because such templates can be implemented on top of any query language.

Queries to the ontology are not limited to question answering. Because the query results are rendered into the wiki pages, they can also serve as a means of dynamically displaying content. Changes in the ontology are then directly reflected on the wiki pages. Possible examples include received awards and lists of works in biographical articles, and index pages, e.g. "all chemists" or "all Nobel prize winners".

So wiki pages can consist of a combination of normal text and ontology data rendered to text. With respect to our defined user groups the following scenario is possible: A contributor creates a new wiki page describing the life of Albert Einstein. In this description he writes about the scientists Albert Einstein worked with. When an expert contributor sees this, he might replace the text formulated in natural language with a semantic query (template). For example he could create a query that yields all scientists who worked with Albert Einstein and further extend the query by a parameter 'country' so that users can query the ontology for all scientists who worked with Albert Einstein and live in a specific country (fig. 8). Such dynamic content depends on the ontology, but it is accessible transparently for all kinds of users. Even a reader with no knowledge about ontologies is able to read the natural language text and select a parameter in a drop down box to get specific information. Fig. 6 shows how a dynamic page is created and fig. 7 shows how a reader sees it.

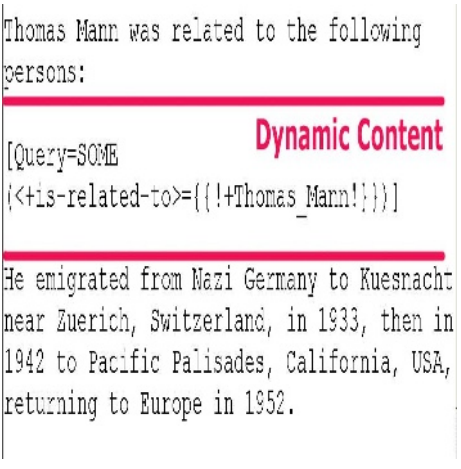

Fig. 6. Queries can be used as dynamic content

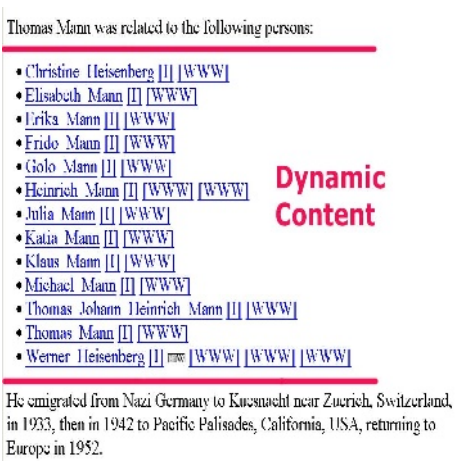

Fig. 7. COW renders dynamic content

7 Case Study: A Biographical Lexicon

We created a small biographical lexicon, containing Nobel Prize winners, their families and coworkers. The natural language content has been taken from the Wikipedia. Important information of the content of biographies like day and place of birth, citizenship and so on have been formalized in the ontology. We created three main classes: `Award`, `Person` and `Location` each with several subclasses. The ontology is populated with 37 instances of `Person`, 12 of `Award` and 52 of `Location`.

The combination of an ordinary wiki together with an ontology leads to an impressive boost of information retrieval. To get a feeling for this extension, we will examine a small example. The user wants to add a new instance called Albert Einstein. Einstein worked together with Max Planck, so he adds the instance `Albert_Einstein` and fills its `worked-with` property slot with the existing instance `Max_Planck`. As the the property `worked-with` is symmetric, COW is able to infer that also Max Planck worked together with Albert Einstein. This fact is also shown the instance view of `Max_Planck`.

The benefit is that we get additional information through the ontology view of `Max_Planck` without editing the instance manually. The ontology not only provides additional information, it also can be used for queries. Queries can directly be implemented into the articles, which allows the representation of dynamic content. If you want to add a list of coworkers in the biography of Albert Einstein, a query can be inserted which updates the list on each page view, using the ontology data. The walk-through shown in fig. 8 summarizes how COW can be used for this kind of application and how users can profit from semantic information.

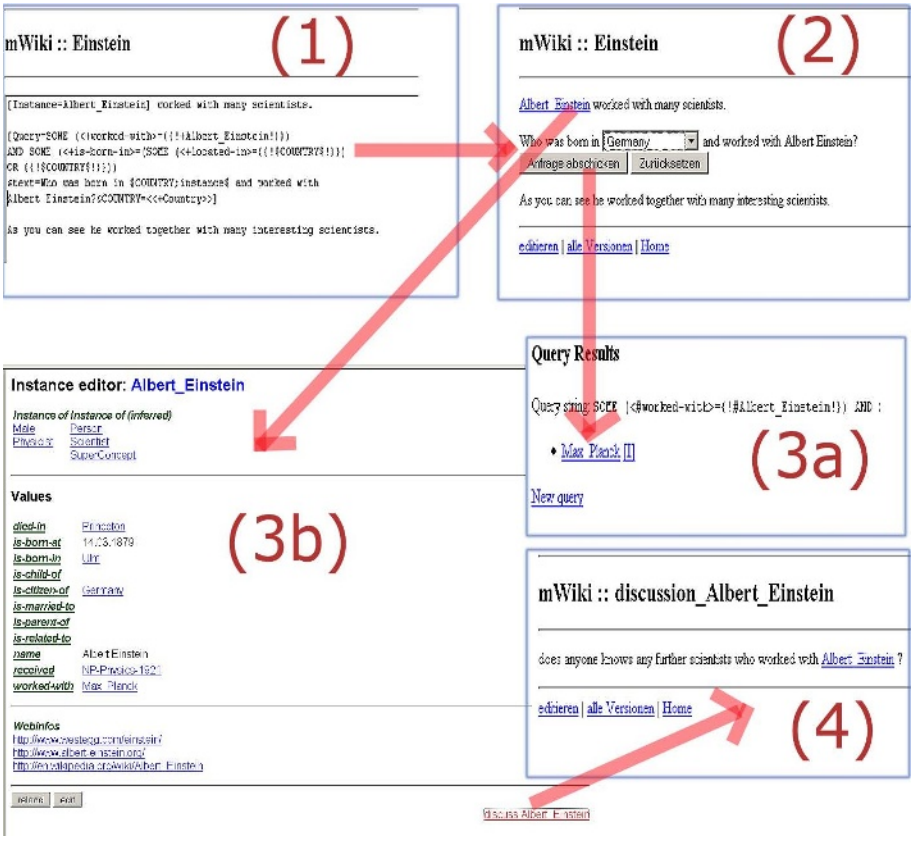

Fig. 8. Query templates are embedded into normal wiki text (1). COW renders this query as a question with forms to be filled out by the user (2). The user has the possibility to execute the query and view the result (3a) or to click on an instance to see further information (3b). Talk pages allow discussions about any entity (4)

8 Conclusion

We presented an architecture for extending wikis for unstructured information to also maintain formalized structured information by *Combining Ontologies with Wikis* (COW). COW has all features of a common wiki and additionally allows the collaborative evolution of an ontology. In contrast to existing semantic wiki systems, it offers an easy-to-use ontology editor and does not confront normal users with ontology data inside the page source code. The editor's smart locking mechanism enables multiple users to work on the knowledge base without unnecessary conflicts; transactions guarantee the ontology to remain consistent. By querying the ontology, dynamic content as part of wiki pages can be created and users can issue queries. Queries can be formulated by means of a formal query language as well as using simple query templates.

In our future work, we will extend COW by input templates, a complementary mechanism to query templates. Input templates will allow inexperienced users to populate the ontology via forms as an alternative to using the web-based ontology editor. Instead of versioning the complete ontology, we will investigate more fine-grained version control policies. Other extensions could be automatic renaming of the entities referenced in a query or a query template when the ontology is changed, the use of ontology engines other than KAON, and queries in emerging standard languages like SPARQL [14] or RDQL [15].

Acknowledgements

We would like to thank Ljiljana Stojanovic and Boris Motik (formerly at AIFB Karlsruhe) for patiently answering our questions regarding KAON, and the three anonymous reviewers whose comments and suggestions helped to improve this article.

References

1. B. Leuf, W. Cunningham. The Wiki Way: Quick Collaboration on the Web Addison-Wesley Longmann, 2001.
2. E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer, G. Stumme, Y. Sure, J. Tane, R. Volz, V. Zacharias. KAON - Towards a large scale Semantic Web. In *E-Commerce and Web Technologies, Third International Conference, EC-Web 2002*, Aix-en-Provence, France, September 2002.
3. T. R. Gruber. A Translation Approach to Portable Ontologies. In *Knowledge Acquisition* Volume 5 Issue 2, 1993.
4. T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. In *Scientific American*, May 2001.
5. R. Tazzoli, P. Castagna, S. Campanini. Towards a Semantic Wiki Wiki Web. Poster Track, *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004.
6. A. Souzis. Building a Semantic Wiki. In *IEEE Intelligent Systems, vol. 20, no. 5*, September/October 2005.
7. M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, R. Studer. Semantic Wikipedia. In *Proceedings of the 15th International Conference on World Wide Web (WWW 2006)*, Edinburgh, Scotland, May 2006.
8. D. Aumueller, S. Auer. Towards a Semantic Wiki Experience – Desktop Integration and Interactivity in WikSAR. In *Proceedings of the 1st Workshop on The Semantic Desktop. 4th International Semantic Web Conference*, Galway, Ireland, November 2005.
9. N. Noy, M. Sintek, S. Decker, M. Crubézy, R. Fergerson, M. Musen. Creating Semantic Web Contents with Protégé-2000. In *IEEE Intelligent Systems*, Volume 16 Issue 2.
10. O. Corcho, M. Fernández-López, A. Gómez-Pérez, O. Vicente. WebODE: An Integrated Workbench for Ontology Representation, Reasoning and Exchange. *13th International Conference on Knowledge Engineering an Knowledge Management (EKAW'02)*, October 2002.

11. S. Auer. pOWL – A Web Based Platform for Collaborative Semantic Web Development. In *Proc. of 1st Workshop Workshop Scripting for the Semantic Web (SFSW'05)*, Hersonissos, Greece, 2005.
12. C. J. Date, An Introduction to Database Systems, Seventh Edition, 2000.
13. B. Parsia, E. Sirin, A. Kalyanpur. Debugging OWL Ontologies. In *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan, 2005.
14. E. Prud'hommeaux, A. Seaborne (editors). SPARQL Query Language for RDF. *W3C Working Draft*, February 2006.
15. A. Seaborne. RDQL – A Query Language for RDF. *W3C Member Submission*, January 2004.