Flavors of KWQL, a Keyword Query Language for a Semantic Wiki

François Bry and Klara Weiand

Institute for Informatics, University of Munich
Oettingenstr. 67, 80538 München, Germany
http://pms.ifi.lmu.de

Abstract. This article introduces KWQL, spoken "quickel", a rule-based query language for a semantic wiki based on the label-keyword query paradigm. KWQL allows for rich combined queries of full text, document structure, and informal to formal semantic annotations. It offers support for continuous queries, that is, queries re-evaluated upon updates to the wiki. KWQL is not restricted to data selection, but also offers database-like views, enabling "construction", the re-shaping of the selected (meta-)data into new (meta-)data. Such views amount to rules that provide a convenient basis for an admittedly simple, yet remarkably powerful form of reasoning.
KWQL queries range from simple lists of keywords or label-keyword pairs to conjunctions, disjunctions, or negations of queries. Thus, queries range from elementary and relatively unspecific to complex and fully specified (meta-)data selections. Consequently, in keeping with the "wiki way", KWQL has a low entry barrier, allowing casual users to easily locate and retrieve relevant data, while letting advanced users make use of its full power.

1 Introduction

Wikis are a popular tool for managing personal and professional knowledge. Wiki content usually consists of simple hypertext documents, that is, web pages which are connected through simple hyperlinks.

In traditional wikis, knowledge is expressed mostly as natural language text and is not directly amenable to automated semantic processing. Therefore, knowledge in wikis can be located only through full-text keyword search or simple (mostly user-generated) structures like tables of content and inter-page links. More sophisticated functionalities such as querying, reasoning and semantic browsing, highly desirable in knowledge intensive, professional contexts such as software development and project management, are not provided.

Semantic wikis promise to provide at least some of these enhancements by relying on so-called semantic technologies, that is knowledge representation formalisms and automated reasoning methods. Semantic wikis add to conventional wikis informal tags and –more or less sophisticated– formal languages for expressing knowledge as annotations to (textual as well as multimedia) wiki pages

that are machine processable. These annotations can range from informal, freely chosen tags to semi-formal tags selected from a pre-defined vocabulary to formal concepts and relationships from an ontology. Thus, semantic wikis are often referred to as "wikis enhanced with semantic technologies" or, after [19], "semantics for wikis". Semantic wikis have also been called "Semantic Web in the small" [7]. On the other hand, semantic wikis are also used as wikis for knowledge engineering, that is, ontology editors of some kind, or after [19], "wikis for semantics". This article uses the term "semantic wiki" in the first sense.

If the data include not only text but also informal and formal semantic annotations, then querying necessarily becomes a difficult task – so at least a conventional wisdom. Conventional web query languages [3] [13] are tools for precise data selection and processing by informed users. As such, they are no candidate query languages for a wiki which is a social application where users should be able to locate content without investing much time or effort.

In line with this reasoning, easier to use query language for semi-structured data aimed at casual users have been developed, often using keywords to express queries. Queries in these languages are usually very simple and imprecise, information retrieval techniques such as fuzzy matching and ranking are consequently used to find documents relevant to the query. While this practice is suitable and convenient for helping casual users retrieve content, the expressive power of such languages is limited. This means that the exact selection of data according to complex criteria is not possible.

KWQL aims to fill the gap between these two kinds of query languages that becomes evident in the social semantic web where selecting data (and being able to do so with precision) is an essential task but where not all users can be expected to be querying experts.

In this article, we introduce KWQL, a query language under development for the semantic wiki KiWi [1]. KWQL refutes the above-mentioned conventional wisdom by relying on the label-keyword query paradigm to allow for selections from full text, from semantic annotations and structure-based selection. Furthermore, following the concept of database views, KWQL offers means not only for selections, but also for "constructions", that is the re-shaping of the selected (meta-)data into new (meta-)data.

The contributions of this article are: (1) Requirements for a wiki query language (2) The syntax of a versatile rule-based label-keyword query language fulfilling these requirements.

The remainder of this article is organized as follows: Section 2 gives and overview over the conceptual model of the KiWi wiki. Section 3 outlines the requirements for a Semantic wiki query language. Section 4 uses example queries to introduce KWQL, Section 5 gives an outlook on query evaluation and functionality to be added in the future. Section 6 presents related work and gives a conclusion.

[1] http://kiwi-project.eu

2 Conceptual Model

In the KiWi wiki, information is represented in the form of content items, links between content items, annotations, meta-data and the structure of content items and fragments.

Content items Content items, the primary unit of information in the KiWi wiki, are composable documents. A content item may consist of a sequence of (included) content items, for example a chapter may consist of a sequence of paragraphs. For reasons of simplicity, content item composition precludes any form of overlapping or of cycles. Thus, content item composition provides a conventional structuring of documents and a content item can be seen as a tree (of content items). Atomic content items may consist of either text or multimedia. A content item is directly addressable by a unique URI. There is no inherent distinction between wiki pages and content items, or rather, by default, all content items are wiki pages. Root nodes, that is, content items that are not embedded in another content items, then have a special status in that they encompass all content that forms a cohesive unit. In this, they can be seen as being alike to a wiki page in a regular wiki.

Fragments Text Fragments are continuous portions of text that can be annotated. Text fragments are useful for –especially collaborative– document editing for adding annotations like "improve transition", "style to be polished" or "is this correct?". For the purpose of this article, it is assumed that fragments can be nested but do not overlap and do not span over content items. Fragments of this kind are generally desirable, but are problematic with respect to query evaluation.

Links Links, that is simple hypertext links as in HTML, can be used for relating content items to each other. Links have a single origin, which is a content item, an anchor in this origin, and a single target, which is also a content item. Links can be annotated.

Annotations Annotations are meta-data that can be attached to content items and links. Two kinds of annotations are available: tags and RDF triples. Tags allow to express knowledge informally, that is, without having to use a pre-defined vocabulary, while RDF triples are used for formal knowledge representation, possibly using an ontology or other application-dependent vocabulary.

Tag syntax and normalization are of significant practical concern but irrelevant to the present paper, thus not addressed in the following.

In the KiWi wiki, annotations can be in the form of –simple or complex– free tags or RDF, but regular users are generally not confronted with RDF, which is considered an enhancement for experienced users. However, querying RDF annotations will be supported in KWQL at a later point.

System meta-data Finally, content items and tag assignments have system meta-data. These meta-data are automatically generated by the system and cannot be edited by the user. System meta-data represent for example the creation and last edit date and the author(s) of a content item or tagging.

In the following, *resources* refers to the basic concepts in the data model –content items, links, fragments and tag assignments (referred to as "tag")– and *qualifiers* refers to properties of resources like their meta-data and tags.

3 Requirements for a wiki query language

Low Entry Barrier A wiki query language should reflect the "wiki way" [21], that is, have a low entry barrier, enabling users to find the information they need without forcing them to undergo extensive training. Searching for wiki pages containing a string, say "query language", should be expressible as this very keyword query, "query language". A more complex syntax might become necessary as more complex selections are expressed – but only in such cases. The transition between simple and complex queries should be smooth and flexible. Conventional web query languages [3] [13] are no candidate query languages for a wiki: With such languages, for example XPath [9] and XQuery [8], even elementary keywords selections turn out to be complex and demanding.

Full Awareness of the Conceptual Model The query language can help express more or less complex selections using any concept of the wiki or combinations thereof. Thus, a content item selection should be possible that refers not only to its textual content but also to the structuring of the content items it includes, to the links from or to the content item, and to its atomic or structured annotations. In short, the query language should be fully aware of the conceptual model.

Answer-closedness The data queried and the query results should adhere to the same data model [26]. That means that query answers are amenable to further queries. This is desirable because no new concept needs to be introduced to represent query answers – the consistency and simplicity of the conceptual model are maintained – and because answer-closedness means that rule chaining can be realized naturally without transformations between different data formats.

Monotonicity As a query gets more complex, it should become more selective. This property, which we call "monotonicity", ensures the intuitiveness of a query language. Monotonicity has limits, of course, primarily when negation is concerned: If "Q" is a query, then the answers to "NOT Q" cannot be expected to be a subset of the answers to "Q".

Construction In a wiki, views are desirable. "View" is understood here in the database sense of pre-defined queries used for making implicit data explicit. If wiki pages contain management information on a project, it might be convenient to generate a wiki page listing the persons contributing to the project from

mentions of personal wiki pages within the project pages. Views require more than selection. They require *construction*, that is, the capability to specify how the selected data are re-organized in a new wiki page, a "view".

Continuous Queries As wikis are tools for work in progress, furthermore in a collaborative setting, tracking wiki data is an essential activity of a wiki user. KQWL's *continuous* queries are a means for the automation of this activity. We call queries continuous that are posed once, and, as the data evolve, are automatically evaluated again and again. For efficiency reasons, continuous queries call for incremental evaluation.

Ranking and Grouping Ranking and grouping are convenient ways to deliver query answers. Inexperienced users tend to pose short, simple queries that may not be very precise and yield many results; grouping enables facetted browsing of query results which helps the user locate the information she is looking for, while the ranking mechanism makes sure that the most relevant results are at the top of the result list.

No Striving for Completeness Completeness is often a desirable property of a query language. Relational completeness ensures that no data can remain unfound in a relational database. We suggest that completeness, however it might be defined, might not be an essential property of the query language of a wiki. Language simplicity is more important and browsing remains an option.

4 KWQL Query Examples

Keywords for full-text search In KWQL, query terms consist of resources which are associated with lists of qualifiers –functioning as labels– and their values, that is, keywords.

```
ci(text:Java title:XML)
```

This query selects content items whose text contains "Java" and which have "XML" in their title. *ci* is the resource, a content item, and text and title are the qualifiers. KWQL has an implicit conjunctive semantics, i.e., if no operator is given, conjunction is assumed. The two qualifier-value pairs here need to separated only by whitespace to indicate that both conditions must be met.

The query `ci(fragment(text:Java))` selects fragments which contain "Java". Since links, fragments and tags are always anchored or contained in a content item, they are considered to be sub-resources of a content item. A query can refer to sub-resources by nesting one resource term inside another, as shown above. Not only content items, but also fragments and links have sub-resources, namely tags and, in the case of fragments, links. Table 1 lists the possible sub-resources for the different resource types.

Query answers in KWQL are always (sets of) content items, ensuring answer-closedness. When no construction is specified, matching content items or fragments are returned. Links and tags are never returned as query answers since

they cannot be easily displayed or understood outside of the context of their associated content item or fragment and to maintain answer-closedness.

The list of matching content items is itself displayed as a content item. The content items returned as answers are either wiki pages or Lowest Common Ancestor (LCA) [15] content items. Wiki pages are defined as content items not embedded in any other content item. Wiki pages are the units of informations visible while browsing, which makes it natural to return them as query answers. On the other hand, the user might only be interested in the parts of a wiki page relevant to his query. In this case, he can select to return only the content item that is an ancestor to all content items in the wiki page that match the query.

Keywords for system meta-data and tags Meta-data are qualifiers (see the list given in table 1). The following query selects content items authored by the user Mary:`ci(author:Mary)`

If both a resource and qualifier are given in the query, the description is assumed to be complete, i.e. sub-resources not stated in the query are not matched when they fulfill the criterion given in the qualifier-value pair.

This strictness of interpretation is required in order to be able to explicitly refer to e.g. a content item's author (`ci(author:Mary)`) but not the author of one of its tags (`ci(tag(author:Mary))`).

Both resources and qualifiers are optional and do not have to be specified in the query, extending the query to all resources or qualifiers respectively.If no qualifier is specified, the query is matched against all qualifier values of the given resource type. "Child", "descendant" and "target" are excluded from this to unintended avoid link traversals. On the other hand, if no resource is given but a qualifier is present, the query matches all resources with the given qualifier(s) and value(s), regardless of resource type.

If only a value but no qualifier or resource are stated, the query is matched on all qualifier values (except those of "child", "descendant" and "target") of all types of resources.

The fact that everything in a KWQL query apart from keywords, that is, qualifier values, is optional, enables more general queries that, at the same time, are easier to construct than fully specified queries.

`author:Mary`

This query, not stating a resource, returns content items of which Mary is the author or that contain tag assignments by Mary.

`tag(Mary)`

Here, no qualifier is given, but the matching is restricted to tag assignments where "Mary" occurs as a qualifier value.

Finally, the most simple KWQL query consists of only a single value, a keyword matched over all qualifiers of content items. It returns all content items where Mary occurs anywhere in the qualifier values of the content items or a contained sub-resource: `Mary`.

"tag" is used to query the tags of content items, links and fragments. The following query selects content items which have a tag containing "Java".

Table 1. KWQL resources and qualifiers

Resource	content item	fragment	link	tag
Qualifiers	URI	URI	target	URI
	author	author	anchorText	author
	created	created		created
	lastEdited	descendant		name
	title	child		
	text			
	numberEdits			
	descendant			
	child			
Subresources	fragment			
	link	link		
	tag	tag	tag	

```
ci(tag(name:Java))
```

Values do not have to be singular but can also be sets as in this query which matches content items that have a tag in whose name both Java and XML occur:

```
ci(tag(name:(Java XML)))
```

The same type of resource can occur several times within another resource, e.g. a content items can contain several links or can be tagged with multiple tags. To specify criteria that two or more distinct instances of a resource should fulfill, the resource and its respective selection criteria have to be given the appropriate number of times. For example, to match a content item tagged with two distinct tags, "Java" and "XML", the query is `ci(tag(name:Java) tag(name:XML))`.

Keywords for Structure KWQL allows for the selection of data based on the structure of documents, that is, content items and fragments using the "child" and "descendant" qualifiers. KWQL does not offer qualifiers for parent and ancestors so to avoid navigational queries, this keeping the language simple. [22] has shown that queries using qualifiers ancestor and/or parent can be expressed without the qualifiers.

KWQL structure qualifiers give rise to recursive data retrieval through a wiki page structure. For example, the following query selects content items which have a tag "Java" and a child content item which has a tag "XML".

```
ci(tag(name:Java) child:ci(tag:XML))
```

Structure qualifiers can thus be seen as links to other content items or fragments and recursive querying as a kind of graph traversal.

Link traversal can be expressed similarly:

```
ci(tag(name:Java) link(target:ci(title:XML)))
```

Here, content items which have a tag "Java" and include a link that points

to a content item that has XML in its title are matched: Note that, although numerous structural queries or link traversals can be nested, no infinite loops can occur. This is because the query is always finite and KWQL does not support Kleene closure.

Connectives and Negation To support more expressive queries, KWQL allows for the combination of several query terms through not only conjunction (which can be explicitly stated as "AND") but also disjunction, expressed as "OR". The following query matches resources authored by Mary or tagged with Java.

```
author:Mary OR tag(name:Java)
```

Query operators are evaluated in order, parentheses are used to specify precedence.

```
ci(text:Java) OR (ci(text:XML) AND ci(title:Java))
```

The unary operator "NOT" is used to express the negation of a query or query term. The following query selects content items which contain "Java" but not "XML" in their text.

```
ci(text:Java AND NOT XML)
```

Variables and Construction Variables and a construction specification enable a fine-grained customization of query results.

```
ci(author:$A title:$T)
```

In this query, the variables $T and A are bound to the titles of content items and their authors respectively. Variable assignments are qualifier-value-like terms where the qualifier specifies the value that the variable name, given in place of a value, is bound to. qualifier-value pairs thus express selection constraints, while qualifier-variable pairs indicate variable assignments.

There are cases where the need to use a qualifier both with a selection constraint and a variable binding arises, for example in conjunction with partial matchings. When all content item tags labels that contain "Java" are to be retrieved and bound to a variable, there is a constraint on the tag labels, but at the same time, the tag labels are to be assigned to a variable.

```
ci(tag(name:(Java -> $X)))
```

"OPTIONAL" is a unary operator used in connection with variables. In the query below, the authors of content items that have "Java" in their text are bound to the variable X. Only if the content items has been assigned at least one tag, additionally the tag name(s) are bound to the variable Y. In the construction, Y can then be used where present, otherwise a value which must be specified by the user (e.g. "no tags assigned") is inserted. The functionality of OPTIONAL could be replicated using two queries, one with the selection of tags and one without, and fusing the results. However, this would be inconvenient for the user and more costly in terms of evaluation.

```
ci(text:Java author:$X OPTIONAL tag(name:$Y))
```

The construction part of a rule creates new data or meta-data from the variables

bound in the query part of the rule. There is thus a clear separation between constraining results and selecting data and processing them for display - the construction part of a rule never selects data, while the query part is not concerned with the presentation of the results.

To maintain answer-closedness, constructions always specify at least one content item. Content items can be created explicitly in the construction, in resource(qualifier:value) syntax. If no content item is specified, the result of the construction is wrapped into a content item automatically.

Given a query that binds titles of content items to T and authors to A, the construct term below creates a list which for an author gives the content items he helped create, presented in a content item with the title "Contents".

```
ci(title:"Contents" text:$A "-" ALL($T,",")))
```

The "ALL" construct serves to collect all possible bindings for the given variable returned by the query. It takes a variable or nesting of variables and construct terms as a parameter. A second, optional parameter indicates the string that is to be used as a separator between the individual variable bindings. The construction below thus yields a name of an author, followed by a dash and a comma separated list of content item titles. "SOME N" can be used in the similarly as "ALL", collecting at most N variable binding instances. N is given as a second parameter.

In addition to re-shaping data, new information can be computed in the construction through the aggregation of data, for example in the form of counting, or determining minima, maxima and averages.

```
ci(title:"Number of Content items" text:$A "-" COUNT($T)))
```

Putting it all together: Rules The query part and the construction part are fused together to form a complete rule.

```
ci(title:"Content" text:$A "-" ALL($T,",")))@ci(title:$T author:$A)
```

Rules in KWQL as presented here are limited to selecting, reshaping and aggregating data into views. By allowing for a broader functionality in the construction part, for example by allowing for the assignment of new tags to existing content items, a simple but powerful language for reasoning could easily be derived.

5 Outlook

Ranking and grouping Ranking should leverage properties specific to semantic wikis to determine a document's relevance such as the number and extent of edits, the approval rating and the author's equity value or relation to the user posing the query. Since a semantic wiki may contain big amounts of data, it is desirable to be able to generate the top-k answers without having to compute query results exhaustively.

The grouping, or clustering, and ranking of results complement each other in helping the user navigate the query results. Facetted browsing of query results, enabled by result clusters, helps the user find the query results most relevant to him by grouping results under several aspects. In the semantic wiki context,

these could be for example tags assigned, but also explicit or implicit social relations between users.

Evaluation of Selections The restrictions imposed on content item and tag composition –cycles in content item structure and in tag orderings are precluded– make an evaluation without memoing possible; however, memoing is necessary in evaluating some RDF/S selections which may be supported in the future. The extent of the RDF query facilities KWQL should offer is still is an open issue.

Evaluation of Rules Both forward and backward rule processing might make sense in a wiki context. Forward rule processing amounts to materializing all views so far specified, thus improving the efficiency of querying and browsing. Functionalities of a wiki such as personalization services might however rely on too large a number of views for full materialization. In such a case, backward rule processing would be necessary.

We currently investigate how rules are used for wiki functionalities, e.g. personalization, aiming at finding a clear-cut separation between rule sets to be evaluated by forward and by backward processing. An evaluation relying on forward processed "system" or "meta rules" implementing a backward processing of application, or "object rules" a la "backward fix-point procedure" [6] or relying on the "magic set rewriting" [4] seems promising.

Evaluation of Continuous Queries Continuous queries are rules that are re-evaluated after each update of the wiki so as to deliver to the user – or service – subscribing to the query an "incremental answer", that is a difference to the previously returned answer(s). The key to computing incremental answers is finite differencing [24]. Finite differencing of a KWQL rule can be pre-computed, resulting in an "incremental" version of the original rule.

The presentation of incremental answers to the user is an issue requiring more research. Both novel answers caused by the last update and invalidated answers "destroyed" by the last update have to be presented to the user. It is an open issue whether simple listings, or more sophisticated presentations are desirable in a wiki context.

6 Related work and conclusion

6.1 Related work

In recent years, research has been undertaken towards keyword querying of structured data, both in databases [17, 16] and for XML and RDF data [31]. Unlike KWQL, the keyword query languages suggested usually do not allow for the creation of views and combined queries over heterogeneous data, although efforts have been made towards combining RDF and XML querying [5, 12]. Further, many of the keyword query languages do not offer a rich syntax, being limited to simple keywords or label-keyword pairs and an implicit conjunctive semantics.

While the need for simple but powerful retrieval of semantic information in semantic wikis has been pointed out [25], current semantic wikis use either simple full-text search [18, 10, 1, 20], allow querying of annotations using a traditional query language like SPARQL, or both [11, 29, 28, 23, 2, 27]; however in the latter case, the two are not integrated but used separately. Embedded queries, a form of views, are possible in several semantic wikis [25].

Semantic Media Wiki [30] uses a query language using a syntax based on a property:value pairs which offers limited capabilities for construction, querying over non-annotation elements of the wiki.

[14] use queries consisting of multiple simple keywords that are translated into conjunctive SPARQL queries to query SMW. Their query language has no connectives, variables or possibilities for customizing views and is limited to querying annotations.

6.2 Conclusion

In this article, we outlined the requirements for querying in a semantic wiki. We presented KWQL, a rule-based query language that is versatile with respect to the expressiveness and, at the same time, simplicity, of queries, and versatile in the respect that it can be used to combine queries over textual data, annotations and structure.

Building on these principles, KWQL enables querying, re-shaping and aggregation of semantic wiki content. An implementation of the core of KWQL as described here is currently underway, while further research is being undertaken to enable the querying of versions and RDF triples and ranking and grouping mechanisms.

Acknowledgements. The research leading to these results is part of the project "KiWi - Knowledge in a Wiki" and has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 211932.

References

1. S. Auer, S. Dietzold, and T. Riechert. OntoWiki – a tool for social, semantic collaboration. LNCS 4273. Springer, 2006.
2. D. Aumueller and S. Auer. Towards a semantic wiki experience – desktop integration and interactivity in WikSAR. In *1st Workshop on The Semantic Desktop*, 2005.
3. J. Bailey, F. Bry, T. Furche, and S. Schaffert. Web and semantic web query languages: A survey. In *1st Reasoning Web Summer School*, LNCS 3564. Springer, 2005.
4. F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursive query processing strategies. *SIGMOD Record*, 15(2), 1986.
5. S. Battle. Round-tripping between XML and RDF. In *Intern. Semantic Web Conf.*, 2004.
6. F. Bry. Query evaluation in deductive databases: Bottom-up and top-down reconciled. *Data & Knowledge Engineering*, 5(4), 1990.

7. F. Bry, J. Baumeister, and M. Kiesel. Semantic wikis. *IEEE Software*, 25(4), 2008.
8. D. Chamberlin. XQuery: A query language for XML. In *ACM SIGMOD Int. Conf. on Management of Data*, 2003.
9. J. Clark and S. DeRose. XML path language (XPath) version 1.0. *W3C*, 1999.
10. A. El Ghali, A. Tifous, M. Buffa, A. Giboin, and R. Dieng-Kuntz. Using a semantic wiki in communities of practice. In *2nd Intern. Workshop on Building Technology Enhanced Learning Solutions for Communities of Practice*, 2007.
11. J. Fischer, Z. Gantner, S. Rendle, M. Stritt, and L. Schmidt-Thieme. Ideas and improvements for semantic wikis. In *The Semantic Web: Research and Applications*, 2006.
12. T. Furche, F. Bry, and O. Bolzer. XML perspectives on RDF querying: Towards integrated access to data and metadata on the web. In *Grundlagen von Datenbanken*, 2005.
13. T. Furche, B. Linse, F. Bry, D. Plexousakis, and G. Gottlob. RDF querying: Language constructs and evaluation methods compared. In *2nd Reasoning Web Summer School*, LNCS 4126. Springer, 2006.
14. P. Haase, D. Herzig, M. Musen, and T. Tran. Semantic wiki search. In *European Semantic Web Conf.*, 2009.
15. D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2), 1984.
16. V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *28th Intern. Conf. on Very Large Data Bases*, 2002.
17. A. Hulgeri and C. Nakhe. Keyword searching and browsing in databases using BANKS. In *18th Intern. Conf. on Data Engineering*, 2002.
18. M. Kiesel. Kaukolu: Hub of the semantic corporate intranet. In *First Workshop on Semantic Wikis: From Wiki to Semantics*, 2006.
19. M. Krötzsch, S. Schaffert, and D. Vrandecic. Reasoning in semantic wikis. In *3rd Reasoning Web Summer School*, LNCS 4636. Springer, 2007.
20. T. Kuhn. Acewiki: A natural and expressive semantic wiki. In *Semantic Web User Interaction*, 2008.
21. B. Leuf and W. Cunningham. *The Wiki way: quick collaboration on the Web.* Addison-Wesley, 2001.
22. D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *Workshop on XML-Based Data Management*, LNCS 2490. Springer, 2002.
23. E. Oren. SemperWiki: a semantic personal wiki. In *1st Workshop on The Semantic Desktop*, 2005.
24. R. Paige. Symbolic finite differencing - part I. In *3rd European Symposium on Programming Copenhagen*, LNCS 432. Springer, 1990.
25. D. Panagiotou and G. Mentzas. A comparison of semantic wiki engines. In *22nd European Conf. on Operational Research*, 2007.
26. S. Schaffert and F. Bry. Querying the web reconsidered: A practical introduction to Xcerpt. In *Extreme Markup Languages*, 2004.
27. S. Schaffert, R. Westenthaler, and A. Gruber. Ikewiki: A user-friendly semantic wiki. In *3rd European Semantic Web Conf.*, 2006.
28. A. Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, 20(5), 2005.
29. R. Tazzoli, P. Castagna, and S. Campanini. Towards a semantic wiki wiki web. In *3rd Intern. Semantic Web Conf.*, 2004.
30. M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic Wikipedia. In *15th Intern. Conf. on World Wide Web*, 2006.
31. K. Weiand, T. Furche, and F. Bry. Quo vadis, web queries. In *Intern. Workshop on Semantic Web Technologies*, 2008.