visKWQL, a Visual Renderer for a Semantic Web Query Language

Andreas Hartl
andreas-hartl@gmx.de

Klara Weiand
klara.weiand@ifi.lmu.de

François Bry
bry@lmu.de

University of Munich
Oettingenstr. 67
80538 Munich, Germany

ABSTRACT

KiWi is a semantic Wiki that combines the Wiki philosophy of collaborative content creation with the methods of the Semantic Web in order to enable effective knowledge management. Querying a Wiki must be simple enough for beginning users, yet powerful enough to accommodate experienced users. To this end, the keyword-based KiWi query language (KWQL) supports queries ranging from simple lists of keywords to expressive rules for selecting and reshaping Wiki (meta-)data. In this demo, we showcase visKWQL, a visual interface for the KWQL language aimed at supporting users in the query construction process. visKWQL and its editor are described, and their functionality is illustrated using example queries. visKWQL's editor provides guidance throughout the query construction process through hints, warnings and highlighting of syntactic errors. The editor enables round-tripping between the twin languages KWQL and visKWQL, meaning that users can switch freely between the textual and visual form when constructing or editing a query. It is implemented using HTML, JavaScript, and CSS, and can thus be used in (almost) any web browser without any additional software.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Design, Experimentation, Languages, Human Factors

Keywords

Wiki, Semantic Web, Semantic Wikis, Keyword Querying, Visual Query Languages

1. INTRODUCTION

The primary goal of the emerging Semantic Web[1] is to make content on the internet more accessible to computerized methods of processing, from querying to automatic reasoning.

Web search, provided by search engines such as Google[1], Yahoo![2] or Bing[3], is used to locate information on the 'traditional web'. Here, queries are lists of keywords, possibly enhanced with simple constructs such as operators for disjunction and negation.

Web query languages like XQuery [4] and SPARQL [5], on the other hand, allow for the precise and targeted selection and transformation of Web data. While these languages are powerful tools, they require their users to be knowledgeable both about the query language itself and the structure and schema of the underlying data. This requirement excludes a large part of the web's user base (and thus the potential user base of the Semantic Web) from the benefits of these languages and the features and functionality they provide.

Visual languages, employing elements like shapes and colors instead of strictly textual syntax, have two advantages over textual languages that specifically benefit beginning users [3]: First, their visual structure can make them easier to learn and understand than textual languages. Secondly, editors for visual languages can support users in the creation of valid queries by providing guidance and preventing editing operations that would result in incorrect queries.

This demonstration introduces visKWQL, a visual query interface for the Semantic Wiki KiWi. visKWQL is not so much a separate query language but rather a visual rendering of KWQL, the keyword-based KiWi query language. It aims at extending the textual query language—which itself has been designed to be easy to use—to achieve two cohesive and tightly integrated querying modi in the KiWi Wiki and enable user-friendly and powerful querying.

The usage of visKWQL to visualize, edit or construct KWQL queries such as `ci(text:Java OR (tag(name:XML) AND author:Mary))` or `ci(author:Mary)` and its interaction with KWQL will be demonstrated in the following.

2. THE KIWI WIKI

Semantic Wikis extend conventional Wikis by the addition of (more or less sophisticated) formal languages for expressing knowledge in terms of machine processable annotations to Wiki pages.

KiWi[4] is a Semantic Wiki with extended functionality in the areas of information extraction, personalization, reasoning, and querying. It supports annotations ranging from informal, freely chosen tags, to semi-formal tags selected from

[1] http://www.google.com

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

[2] http://www.yahoo.com

[3] http://www.bing.com

[4] http://www.kiwi-project.eu, showcase available at http://showcase.kiwi-project.eu/

a pre-defined vocabulary, to formal concepts and relationships from an ontology.

The basic conceptual units in KiWi are content items, text fragments, links and tags:

Content Items are the basic units of information in KiWi, they correspond roughly to Wiki pages in other Wikis. However, content items can also include other content items. The nesting of content items then forms a tree structure. Each content item has a URI through which it is accessible and uniquely identifiable. Content items can contain fragments, links and tags.

Text Fragments are continuous portions of text within a content item that users can define and annotate. They can consist of a word, a sentence, or any other section of text.

Links in KiWi behave similarly to hypertext links. They have an anchor (a fragment) and a target content item. Links can be annotated.

Tags are annotations that can be set by any user, but can also be created by the system through automatic reasoning.

Each of the four conceptual units is further associated with meta-data, like its author, creation time and versioning information.

3. KWQL IN A NUTSHELL

KWQL[5] [2], the KiWi Query Language, is a rule-based query language based on the label-keyword paradigm. It aims at combining a low entry barrier with powerful querying, in order to accommodate users with varying levels of expertise. KWQL combines the ease of use of keyword search with advanced features and capabilities as used in traditional query languages.

KWQL can query the elements of the conceptual model, their properties like textual content and meta-data as well as the structure of content item and fragment nestings. Basic elements of a query are *resources*, that is, the units of the conceptual model discussed above, and *qualifier terms*, label-value pairs referring to a resource's property types and their values. A qualifier label can be user-created data like `text` or `title`, which refer to the textual data and the title of, e.g. a content item. Qualifier labels can also refer to types of system-generated metadata or to structural properties like `child` or `descendant`, which represent direct or indirect children in a nesting of content items or fragments.

A basic KWQL query term, a *resource term*, is of the form `resourceType(qualifierLabel:value)`. Resource types and qualifier labels are optional, respectively extending the query to all resources or qualifier values when omitted. The simplest query in KWQL thus consists of a single keyword, corresponding to a search over all qualifier values of resources of all types. Further examples of KWQL queries are given in table 1.

KWQL supports conjunctions, disjunctions and negations of values or qualifier or resource terms; parentheses can be used to indicate precedence. Resources can be nested. For example `ci(tag(KiWi))` returns all content items which are tagged with "KiWi".

In addition to data selection, KWQL also allows for the creation of rules to reshape and aggregate query results into new data. Rules consist of a query and a construction part. The query part is equivalent to a regular query, but can additionally include variable bindings. The construct part of a rule can contain operators to extract values from bound variables. For example, the rule `COUNT($A) @ ci(author:$A title:"An introduction to KiWi")` would return the number of authors of the Wiki page titled "An introduction to KiWi". The query part of the rule, on the right hand side of '@', searches for content items with the given title, and binds the item's author qualifier to the variable A. In the rule's construct part, the operator $COUNT$ is used to extract the number of values bound to A, which equals the number of authors of the Wiki page.

Table 1: Examples of KWQL queries

`Java` Select documents containing "Java" in any of the qualifiers' values, e.g. text, title, tag label or author names
`ci(author:Mary)` Select documents authored by Mary
`ci(text:Java OR (tag(name:XML) AND author:Mary))` Select documents that either have "Java" in their text or that have the tag "XML" and were authored by Mary
`ci(tag(name:Java) link(target:ci(title:Lucene) tag(name:uses)))` Select documents with the tag "Java" that contain a link tagged "uses" to a document with the title "Lucene"
`ci(title:Contents text:($A "-" ALL($T,","))) @ ci(title:$T author:$A)` Retrieve the titles and authors of all documents and display them in a new document

4. VISKWQL

visKWQL provides a visual alternative to textual KWQL. It fully supports KWQL in that every KWQL query can be expressed as an equivalent visKWQL query. Further, in order to avoid introducing additional constructs and thus additional complexity, visKWQL stays as close as possible to the textual language in its visual representation.

Figure 2 shows the interface of the visKWQL editor.[6] The numbered elements allow to select query building blocks (①), save and load queries (②) and construct queries (③). The editor further displays tooltips to guide the query creation process (④) and shows the textual version of the current query (⑤). The latter can also be used to visualize and edit textual queries, as will be explained below.

4.1 Visual Formalism

visKWQL uses a form-based approach, in which all KWQL elements, including resources, qualifiers, and operators, are represented as boxes, and resource-value or qualifier-value associations are represented as nestings (see Figure 3 for an example). Boxes consist of a *label*, in which the name of the represented KWQL element is included, and a *body*, which can hold child boxes.

This approach has several advantages: it stays close to KWQL's textual structure, keeping visKWQL simple and making it easy to translate between the two representations; it also lends itself well to rendering in HTML.

[5] An implementation of KWQL is part of the latest KiWi prototype.

[6] A demo is available at http://pms.ifi.lmu.de/visKWQL.

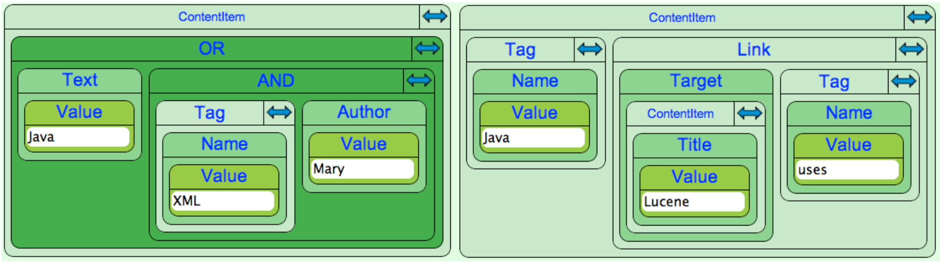
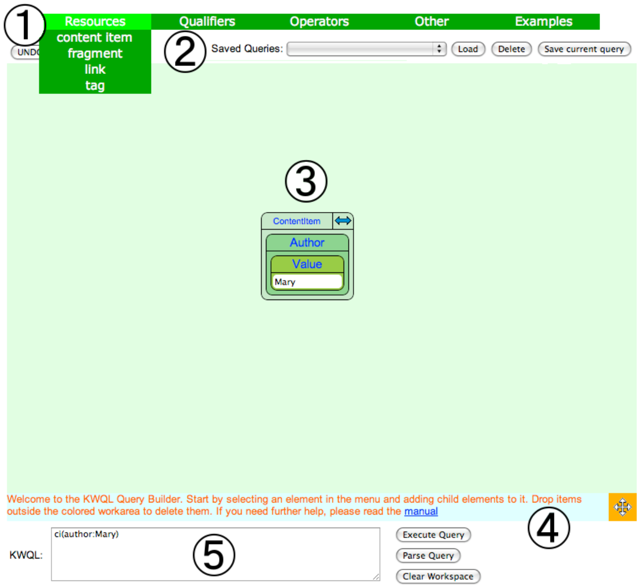


Figure 1: The third (left) and fourth (right) example from table 1 as visKWQL queries



Welcome to the KWQL Query Builder. Start by selecting an element in the menu and adding child elements to it. Drop items outside the colored workarea to delete them. If you need further help, please read the manual

Figure 2: The visKWQL editor

To distinguish between different kinds of elements like resources, qualifiers and operators, each of them is given a different color. In accordance with the KiWi color scheme, visKWQL as integrated in KiWi uses different shades of green for the different elements, ranging from light green for resources to Kelly green for operators.

4.2 Round-tripping

One of the key features of visKWQL is round-tripping, to achieve a tight coupling between KWQL and visKWQL. This property is implemented via a parser for the textual KWQL language written using the parser generator ANTLR[7] and included in the visKWQL editor.

Whenever the user makes a change to the visual query, the change is immediately represented in the textual version. The textual query can further be edited and parsed by the system to display it in its visual form.

This allows the user to make changes in the representation of his choice at any time during the query construction process, to import KWQL queries easily into visKWQL, and has the additional benefit of teaching the user KWQL while he experiments with the visKWQL editor.

4.3 visKWQL Queries in Practice

When the editor initially loads, the workspace is empty. The user can start either by writing or pasting a textual KWQL query into the text box (⑤ in Figure 2), by selecting an element from the menu bar (③), or by loading a query saved earlier (②).

To create the query shown in the workspace of Figure 2, the user simply clicks on "Resources" in the menu bar and selects "content item". A new content item box will appear in the workspace. As only content items authored by Mary should be matched, the qualifier "author" must next be selected from the menu and dragged into the content item element. Finally, the value "Mary" must be entered. The text box displays the textual version of the query, `ci(author:Mary)`.

Further children can be added to the content item element by clicking the blue double arrow in the upper right corner of the box. This will increase the size of the box to make room for a further child, for example a qualifier box with the value "XML" to limit the query answers to content items authored by Mary and containing "XML" in the text.

If the text is altered, for example to reformulate the query to `ci(title:Mary)` and find content items where "Mary" appears in the title, and "Parse query" is clicked consequently, the visKWQL query is altered to reflect this change. The same result can be achieved by dragging the qualifier box out of the content item box and replacing it with a "title" qualifier. This can be done by selecting it from the menu and dropping it into the box representing the content item. Type switching offers a convenient shortcut for this type of procedure: when a box is dropped on the label of another syntactically equivalent box, their types are swapped. This allows for a quick change of elements without the need to move child boxes contained in them. In the case of text boxes, their values are exchanged. The "title" qualifier could, for example, simply be swapped with the "author" qualifier without the need to re-enter the value, "Mary".

A query can be saved for future use by clicking on "Save current query". A name for the query can be entered and will consequently appear as one of the choices in the drop-down menu. Currently, queries are saved on a per user basis.

A query can be deleted by simply dragging it out of the workspace, that is, the light-green colored area. Any change to the query, including the addition and deletion of elements, can be reverted using the "undo" button in the upper left corner of the editor.

The example query above is relatively simple and its visKWQL representation does not take up a big portion of the workspace. Queries however require increasing amounts of screen space as they get more complex. To help the user

[7] http://www.antlr.org

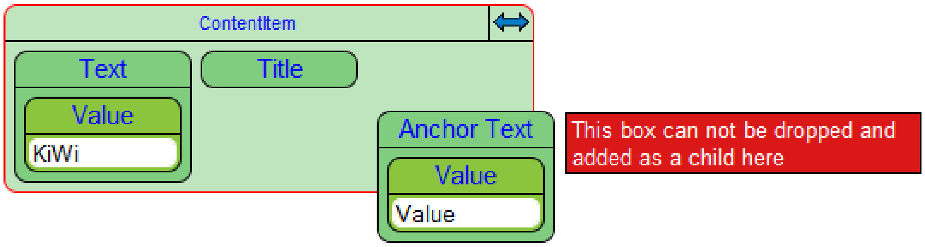


Figure 3: Information hiding and error prevention

maintain a clear mental picture of his query, boxes can be *folded*: when the user clicks on the label of a box, its body and all children contained therein are hidden from view. Figure 3 shows an example in which the body of the "Title" box has been hidden. The workspace can further be resized by clicking the orange box with the white arrow symbol in the bottom right corner and dragging until the workspace has reached the intended size and proportion.

Evaluation of a query is finally triggered by pressing the button "execute query" next to the KWQL text box.

Additional visKWQL queries corresponding to examples three and four in Table 1 are shown in Figure 1

4.4 User Guidance

User support in visKWQL is provided via tooltips, error prevention, and error and problem display and correction.

Tooltips: A text area below the workspace displays an explanation of the KWQL element represented by the box currently under the mouse cursor. Tooltips are also displayed for menu items.

Error Prevention: A large number of syntactic errors result from invalid box nestings, and can be actively prevented by the editor during drag and drop actions. When a box is being dragged, the system continuously checks the validity of a child inclusion or a type switch with the box underneath it. When a box may be dropped in its current location, the parent element is surrounded by a green border, and a green colored tooltip appears next to the dragged box.

If dropping the box in its current location would result in a syntactic error, the border of the box underneath it is colored red, and a tooltip informs the user that he may not drop the box (see Figure 3). A box dropped in an invalid place is returned to where it was picked up.

Error Reporting and Correction: Some errors cannot be prevented during editing. These include variable names or values containing invalid characters, empty strings, misplaced operators and references to undefined variables.

After every user action, the query is checked for such errors. When an error is found, the label of the node is colored red and a tooltip indicating the error is displayed next to it.To make these errors easy to locate within the query, even if the erroneous node is currently hidden, the labels of all its parent boxes will also be colored red, and display a tooltip that a child box contains an error.

Errors that are less severe and can be corrected automatically, like empty boxes, cause the box label to be colored orange. A tooltip and a message below the workspace inform the user about the source of the problem. In addition, the system will internally correct the query, e.g., by removing empty boxes, so that the textual query is still valid.

4.5 Implementation

KiWi is a web application implemented using the JBoss Seam web programming framework and running on a JBoss[8] server.

Consequently, visKWQL can be used from within a web browser without the need to install additional software or browser plugins. The visual interface runs client-side and is implemented in DHTML, using HTML for the static page layout of the editor, JavaScript for the dynamic and interactive parts, and a CSS stylesheet to specify the graphical presentation. The editor can communicate with the KiWi server through AJAX to save or load queries or initiate query evaluation.

The visKWQL editor was designed to make it easy to change its appearance and localization. Consequently, all its graphical data is contained in one CSS file, while all text displayed to the user is located in one single text file, and no values are hard-coded. Changing the color scheme or language of the visKWQL editor is as easy as editing one of the files.

5. 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.

6. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[2] F. Bry and K. Weiand. Flavours of KWQL, a keyword query language for a semantic wiki. In *Proceedings of SOFSEM 2010*, 2010.

[3] T. Catarci, M. Costabile, S. Leviladi, and C. Batini. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[4] D. Chamberlin. XQuery: A query language for XML. In *ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[5] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF (working draft). Technical report, W3C, March 2007.

[8] http://www.jboss.org