QuiKey – an Efficient Semantic Command Line

Heiko Haller

Forschungszentrum Informatik (FZI), Germany
heiko.haller@fzi.de

Abstract. QuiKey is an interaction approach that offers interactive fine grained access to structured information sources in a light weight user interface. It is designed to be highly interaction efficient for searching, browsing and authoring semantic knowledge bases as well as incrementally constructing complex queries. Empirical evaluation using a comparative GOMS Analysis and a user study confirm interaction efficiency.

1 Introduction

Many knowledge management systems, especially those which rely on highly structured information and meta data being entered and maintained by users, fail because users do not make this additional effort. This may be one of the reasons why semantic technologies have, so far, not found widespread use in knowledge management systems, although semantic meta data would undoubtedly improve findability, interoperability and, in general, automated processing of information and knowledge items. QuiKey is a user interface concept that provides a light-weight, generic tool for searching, browsing and editing structured information in a fine-granular way. Additionally, and in the same interaction paradigm, QuiKey allows the construction of simple semantic queries as well as combining these simple queries to more complex ones in a step-by-step manner. QuiKey's main design goal is efficient interaction. It is targeted to cover the following use cases: targeted search of information (e.g. someone's phone number or someone's girlfriend's e-mail address), fast information entry (e.g. adding a new contact and linking her to an existing project), text search, formulating simple queries (e.g. all members of a certain project), set-based browsing [1] (explained in Section 2.2) and incrementally constructing complex queries (e.g. getting a list of members of projects financed by the EU that live in Portugal).

This paper describes the interaction design of QuiKey, and its current prototypical open source implementation together with a twofold evaluation study to substantiate its claims.

2 Design

The primary design goal of QuiKey was to make interaction as efficient as possible. Everything should be done with the least interaction effort necessary. While interaction efficiency is generally desirable, it is even more crucial for mobile

devices where typing is usually cumbersome. The secondary design goal was, to have a minimalistic screen design that is also suitable for constrained screen space like in mobile devices.

The following principles have guided the interaction design of QuiKey:

- Everything can be done with the keyboard alone. While mouse interaction is always possible, it is never required. This avoids unnecessary costly switches between input media (also referred to as "homing" in user interaction literature [2]).
- There is only one mode for everything. Searching, browsing, authoring and querying is all done in the same consistent way of interaction.
- Short feedback cycles to reduce error-proneness. All parts of a complex interaction (items, relations, query operators and such) are implicitly or explicitly selected from lists of existing things, and the structure of the current operation is reflected visually. Like this, misspellings or syntax errors are greatly avoided and if they occur they are easy to notice.

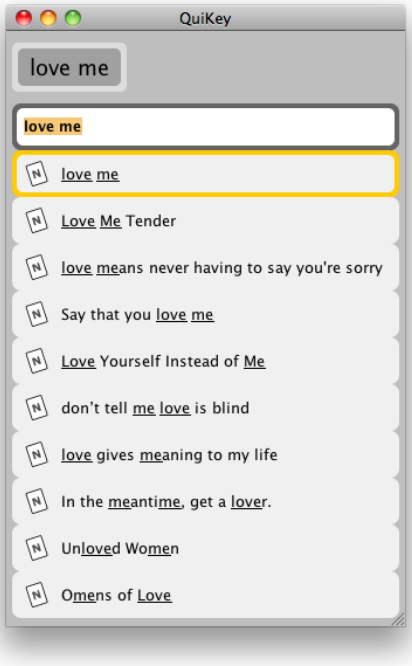

Fig. 1. Text Search: A number of items is shown that match the search string "love me".

QuiKey is organized around the notion of *parts*. A part can be an existing item, a relation, a new text string or a command. Depending on the types and order of the parts entered, it is decided what action to take. The following are the main functionalities of QuiKey. As explained below, they are tightly interwoven.

2.1 Text Search

The simplest functionality that is also usable without any understanding of the structure of semantic knowledge models, is full text search. While search terms are entered, a list or ranked results is displayed based on a set of matching rules explained below. The best hit is preselected, so any item can be addressed by mere text entry without the need to use any special keys for syntax elements or selection. Of course, other items can also be selected via arrow keys or mouse. The following ordered list of ranking principles determines the ranking of text search results in QuiKey:

– a perfect match between search string and item text is best
– matching the beginning of an item is better than matching it anywhere else
– matching full words is better than matching a prefix only
– matching prefix is better than matching an arbitrary substring only
– matching the complete, coherent search string is better than matching single search words separately
– matching several search words in the right order is better than random order

This text search functionality is used throughout QuiKey wherever an item, relation type or such needs to be identified and it is one of the reasons that make QuiKey efficient. For example, to bring the item "Michael Jackson" to the top position out of 43270 named entities, it suffices to type "m jac".

2.2 Browsing

Starting with an item selected by text search, a user can navigate the knowledge base through its graph structure hop by hop by hitting the tab key.

When an item is jumped to, all corresponding statements (triples) about this item are displayed – sorted by relation types. When an item and a relation are selected (e.g. Madonna→has genre)[1], all statements matching this pattern are displayed as in Figure 2. From there, any statement can be selected to jump to the target object of the statement (in our example a particular album). Like this, a user can browse from entity to entity with the pattern *item*→*relation*→*item*→*relation*→*item*... (e.g. `Madonna`→`has album`→`Like a Virgin`→`has genre`→`Pop music...`).

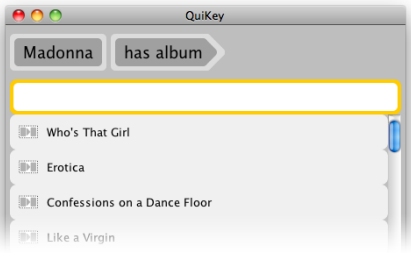

Fig. 2. Browsing: A list of all albums of madonna is shown after selecting the item "Madonna" and the relation type "has album".

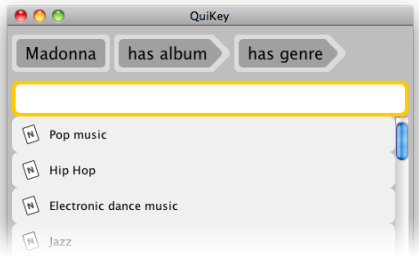

Fig. 3. Set-based browsing: A list of all genres of all albums of madonna is shown.

Set Based Browsing It is also possible to browse from sets of items to related sets by using the pattern *item*→*relation*→*relation*... For example, `Madonna`→`has album`→`has genre` would give a list of all genres of all albums of Madonna as in Figure 3.

[1] The arrow symbol →is used here to separate input parts, which is done with the tab key in QuiKey.

2.3 Queries

Constructing complex, possibly nested, queries over structured or semi-structured data with a text-based query language (like querying an RDF [3] graph with SPARQL [4] or formulating ASK Queries in Semantic MediaWiki [5]) is difficult: every slight syntax error or misspelling makes the whole query fail or (worse) return unintended results. And there is usually no feedback as to where the error lies because complex queries are formulated and evaluated as a whole only. QuiKey tackles these two common problems:

(1) Misspellings and syntax errors are largely avoided because instead of requiring the user to write a whole query in some complicated syntax, in QuiKey, simple queries are constructed by browsing interactively, selecting from existing items and without the need of syntactical characters.

(2) To facilitate modular construction of complex queries in a step-by-step manner, each query can be saved and referred to as a special query item:

Saving Queries In QuiKey, like in facetted browsing, the border between browsing and query construction is blurred. In fact, the two above browsing examples already form semantic queries. Such simple queries can be persisted by inserting a name, under which the query should be saved, in the respective position of the pattern as a placeholder – prefixed by a question mark. For example: `Madonna→has album→?Madonna's albums` would save a new query item named "Madonna's albums" that represents all of madonna's albums. The placeholder does not need to be in the last position, e.g., the pattern `?Madonna's Genres→is genre of→is album of→Madonna` would save a query item representing all genres of all albums of Madonna.

Complex Queries More complex queries can be constructed by combining existing saved queries with the logical operators `and` and `or` (e.g. Madonnas's genres→and→Jacko's genres). Since also complex queries can be saved, more complex queries can be constructed step by step out of existing ones.

2.4 Authoring

With QuiKey, knowledge bases can be altered in the following different ways:

Adding Items To add a new text item to the knowledge base, it is enough to just type the text and press enter.

Adding Statements To make statements about existing items, the statement can be entered in a subject→predicate→object pattern, separated by tab-keys. So, for example, `Michael Jackson→has album→This Is It` would add this statement to the knowledge model. Only that the user would not even have to type in the whole labels because parts that are already known can be chosen from the suggestions list while typing in auto-completion manner. So, for this example, it is actually enough to type in `m jac→h albu→This Is It`.

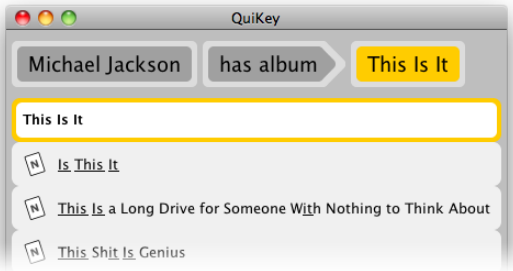

Fig. 4. Adding Several Objects: A new item "This Is It" is about to be created together with the statement that it is an album of Michael Jackson.

Adding Items and Statements Together If not all three parts in such a statement are known objects, the respective items or relation types are also added to the knowledge base. So, in the above example, if "This Is It" is not a known item, it would directly get created, together with the statement that it is an album of Michael Jackson. To avoid accidentally creating new items instead of reusing existing ones, the respective part is highlighted in yellow during interaction, i.e. before the action is executed. This can bee seen in Figure 4.

3 Implementation

QuiKey was initially developed as part of the semantic desktop project nepomuk[2]. For more information on semantic desktop systems in general, see [6] and [3]. While the QuiKey approach could be used with any kind of graph-based knowledge base, the current implementation uses a back-end and data model called CDS (Conceptual Data Structures) that has also been developed in the nepomuk project. The Java based CDS-API also features an in-memory inverted sub-string index, that serves QuiKey's text search functionality. The preliminary set of matching items is then ranked by quikey according to the matching rules described in Section 2.1. For performance reasons, the search depth of QuiKey's matching rules can be adjusted by the user. However, even with all rules enabled, text searches with up to 4 search words are executed well below one second on a knowledge base with 43270 items on a 2.4 GHz Intel core duo processor – with up to 3 search words, results are usually perceived as instantaneous. Since the expressiveness of neither CDS nor QuiKey's queries exceeds EL++[7], there could also be optimized implementations that scale to very large knowledge bases without slowing down user experience. For more information about CDS, see [8].

QuiKey's current open source implementation is based on Java/Swing. Today, QuiKey is deployed to complement iMapping, a visual knowledge workbench [9] that is also based on CDS and developed in nepomuk. For more information about iMapping and to get the latest version of both iMapping and QuiKey, see `http://imapping.info/`.

[2] `http://nepomuk.semanticdesktop.org`
[3] `http://semanticdesktop.org`

4 Related Work

4.1 Quicksilver

Quicksilver[4] by Nicholas Jitkoff is a kind of advanced application launcher for the Mac that has gained a lot of popularity due to its versatility and efficiency. QuiKey is mainly inspired by quicksilver. It is the attempt to adapt and transfer quicksilver's highly efficient interaction paradigm to the semantic desktop. However, QuiKey differs from Quicksilver in several ways: (a) While both allow to browse structured information models, Quicksilver is for finding and acting upon certain desktop objects. QuiKey is a generic authoring and query tool for graph-based knowledge bases. (b) Quicksilver matches the letters of the search string entered in the exact order only. It does not distinguish separate search words and will e.g. not match "Ontology Web Language" to "Web Ontology Language".

4.2 Parallax

Parallax by David Huynh [1] is probably the most convenient user interface to date to explore large amounts of structured data. Parallax is an experimental front end to Freebase[5], a website that offers a large amount of open structured data. A video that is also embedded in the parallax home page[6] nicely explains the benefits of semantic search in general and parallax in particular. It features the notion of set-based browsing, where navigation takes place from one set of things to another related set of things (e.g. from Michael Jackson's albums to their genres). It partly addresses the same use cases as QuiKey (querying large graph based knowledge bases through navigation), and features a visually much richer user interface (many navigation options per view, picture content, thumbnail previews, et c.).

5 Evaluation

The main claim of QuiKey's interaction design, to be highly interaction efficient, has been evaluated in two phases: First, in a comparative interaction analysis according to the KLM-GOMS method, where QuiKey has been measured against Semantic MediaWiki and parallax. Second, in a user study, where actual interaction times have been measured for QuiKey in order to validate the outcomes of the GOMS analysis. To that end, a set of tasks has been defined, by means of which the respective tools could be compared.

[4] Original homepage: `http://blacktree.com/?quicksilver`
[5] `http://www.freebase.com/`
[6] `http://www.freebase.com/labs/parallax/`

5.1 Tasks and Data

The goal that was chosen as a basis for the evaluation, was to construct a conjunctive query that yields the answer to the question *Which musical genres do Madonna and Michael Jackson have in common?*. This goal fulfills the following requirements: (a) It can be decomposed into single steps that cover a wide range of QuiKey's functionalities (searching for items, browsing their properties, constructing simple and complex queries and adding new items). (b) It is comparable to other tools that offer similar functionality. (c) It is easy to understand because the musical domain is common knowledge. (d) A large amount of structured data is publicly available. In fact, an export from freebase of the music domain has been taken (artists, albums, genres etc.). It was filtered down to yield a data set of 26115 items that was highly interconnected to allow for complex semantic queries but small enough to run smoothly with the current CDS back end which was designed to handle personal knowledge management data fast and in memory rather than large imported data sets.

This goal can be broken down to the following sub-tasks:

1. Find out what albums Madonna has made. (text search, browse)
2. Find out what genres these albums have. (browse set)
3. Save this as a persistent query. (save query)
4. Do the same (1–3) for Michael Jackson.
5. Intersect these two saved queries. (construct complex query)

These sub-tasks have been used for comparison in the GOMS analysis and the user study.

5.2 GOMS Analysis

KLM-GOMS is a method developed by Card, Moran & Newell [2] to estimate the time it takes a user to complete simple interactive tasks using a keyboard and mouse[7]. This was done for each of the three tools. Once for the most efficient way the tool could theoretically be used for the task and once for the typically expected way (e.g. query code was formatted with white space, query names were more verbose ("Jacko genres" instead of "MJgen") and search words were spelt out instead of only to the point necessary.

The tools that were chosen for comparison are Semantic Mediawiki[8], [5] and parallax[9], [1]: Semantic MediaWiki with its ASK query language is probably the most widely used semantic knowledge modeling tool that also allows to construct complex and persistent queries. Parallax is probably the most convenient user interface to date to explore large amounts of structured data.

[7] for an overview see `http://en.wikipedia.org/wiki/KLM-GOMS`
[8] `http://semantic-mediawiki.org/`
[9] `http://www.freebase.com/labs/parallax/`

Results The resulting estimates of interaction time needed are shown in Table 5.2. Unfortunately, in parallax it appears not to be possible to intersect existing queries or sets. Saving a current query also seems not to be possible. However a straight forward way how this would be done in parallax' interaction paradigm is apparent and so this way was guessed as an approximation. As can be seen

Table 1. KLM-GOMS analysis for the sub tasks in comparison. Typically expected interaction and theoretically minimal paths of interaction are computed separately. Overall results for QuiKey are bolded for comparison with Table 2.

	SMW		**parallax**		**QuiKey**	
task	typical	minimal	typical	minimal	typical	minimal
one-time overhead per query	8.2	8.2	0	0	0	0
elementary query Madonna	13.2	13.0	10.8	10.8	7.5	6.1
elementary query M. Jackson	16.4	16.1	13.1	13.1	8.1	6.4
increment to chain query (x2)	18.3	17.2	7.2	6.6	3.5	2.9
increment to save (x2)	19.4	19.4	8.2	6.5	7.5	5.8
intersection query	21.0	21.0	n/a	n/a	10.7	7.6
overall time w/o intersection	113.2	110.4	54.8	50.3	**37.5**	**30.0**
overall time incl. intersection	134.2	131.4	n/a	n/a	**48.2**	**37.6**

in Table 5.2, QuiKey is theoretically more interaction efficient than any of the two compared tools on any of the tasks. This means that, learnability, interaction styles and error-pronenes aside, it is theoretically possible to complete these tasks faster in QuiKey than in the other tools. What remains to be shown is that QuiKey can actually be *used* in this efficient way by real users:

5.3 User Study

In order to determine whether QuiKey can be used by real users as efficiently as it is designed to be, we let 16 testers perform the above defined set of tasks. All testers were familiar with semantic technologies in general. 3 of the testers were female, 13 male. The age of the testers ranged from 23 to 36. 14 of the 16 testers had never used QuiKey before and only one was already familiar with it.

Each tester went through the following process:

1. Short introduction to QuiKey: basic functionalities and interactions that are needed for the tasks were explained and demonstrated. Testers were asked to try out the interactions and explore the model and the tool until they feel confident in using it. (This took up to 8 minutes.)
2. A screen capture tool was started, that recorded screen content, audio, key strokes and the users via screen cam.
3. Testers were then asked to carry out the set of tasks. Instructions were given sequentially one by one, whenever the previous step was completed.

Later, keystrokes and interaction times were determined from the captured video with a precision of 18 frames per second.

Results The mean number of keystrokes and interaction times are listed in Table 2 along with their confidence intervals[10]. The last column lists the minimal interaction times any user had needed to complete the task. Since these values come from different users they do not add up to the overall times. The overall minimum times instead reflect the time of the overall single fastest user. These minimum times are included in the table because they prove for each task how fast it *can* actually be done. Comparing the bolded values shows that the overall

Table 2. Mean times and key strokes needed for sub-tasks. (confidence intervals for p=.05%)

	keystrokes		**measured time**		
task	mean	±CI	mean	±CI	minimum
elementary query Madonna	10.8	±1.1	9.2	±1.2	4.8
elementary query M. Jackson	12.7	±1.9	8.9	±1.5	4.3
increment to chain query (x2)	7.6	±0.4	7.1	±0.8	3.2
increment to save (x2)	18.7	±2.6	8.7	±1.2	2.5
intersection query	20.1	±2.7	16.0	±3.1	8.1
cum. interaction time w/o intersection			**50.1**	±6.4	**31.8**
cum. interaction time incl. intersection			**66.1**	±9.1	**38.9**

GOMS estimation of minimal interaction times were very close to the actual minimal times. This supports the GOMS estimations in general. Mean interaction times were somewhat longer than estimated for typical use. This may well be due to the fact that most of the testers were first time users. Also, during user testing it was noticed that search words were often spelt out completely instead of only to the extent needed (e.g. "Michael Jackson" instead of "m jac"). This is because users do not check results after every keystroke while typing known words. However in situations where keyboard interaction is more costly, like on mobile devices or touchscreens, for motion impaired users, or for long or unfamiliar words, the use of shorter search strings becomes more relevant.

6 Conclusion

Some of the claims, why the interaction approach of QuiKey is beneficial may be convincing by argument and some may be hard to prove in a lab study. But the the fundamental claim of interaction efficiency has been well confirmed through GOMS analysis and the user study.

Additional functionalities that are subject of future work include

[10] `http://en.wikipedia.org/wiki/Confidence_interval`

– displaying short explanations about what a pattern means during interaction interacting, before it is executed, so the user knows what is going to happen and giving unobtrusive confirmations on actions completed,
– a way to interactively construct complex queries from scratch without the need to name and save intermediate queries,
– improving matching rules during use and automatically learning shortcuts for frequently used items,
– more expressive query operators including negation,
– history and undo
– using QuiKey as an additional front-end to Semantic MediaWiki
– developing a mobile version, where some of QuiKey's minimalist interactions have a bigger benefit, because typing is more costly.

Acknowledgements

This Work has partially been financed by the European Commission in the projects NEPOMUK (IST-FP6-027705) and MATURE (IST-FP7-216356). Special thanks go to Florian Simon for the implementation work of QuiKey.

References

1. Huynh, D., Karger, D.: Parallax and companion: Set-based browsing for the data web. Technical report, Metaweb, MIT (2009) Available online: `http://davidhuynh.net/media/papers/2009/www2009-parallax.pdf`.
2. Card, S.K., Moran, T.P., Newell, A.: Psychology of Human-Computer Interaction. Lawrence Erlbaum (1983)
3. Manola, F., Miller, E.: Resource Description Framework (RDF) primer. W3C Recommendation (2004) Available at `http://www.w3.org/TR/rdf-primer/`.
4. Prud'Hommeaux, E., Seaborne, A.: Sparql. W3C Candidate Recommendation (June 2007)
5. Krötzsch, M., Vrandecic, D., Völkel, M., Haller, H., Studer, R.: Semantic wikipedia. Journal of Web Semantics **5** (SEP 2007) 251–261
6. Sauermann, L., Kiesel, M., Schumacher, K., Bernardi, A.: Semantic Desktop. In Blumauer, A., Pellegrini, T., eds.: Social Semantic Web, Springer-Verlag, X.media.press (2009) 337–362
7. Krötzsch, M., Rudolph, S., Hitzler, P.: Complexity boundaries for horn description logics. In: Proceedings of the 22nd AAAI Conference on Artficial Intelligence, Vancouver, British Columbia, Canada, AAAI Press (2007) 452–457
8. Völkel, M., Haller, H.: Conceptual data structures for personal knowledge management. Online Information Review **33**(2) (2009) 298–315
9. Haller, H., Abecker, A.: imapping – a zooming user interface approach for personal and semantic knowledge management. In Toms, E., Bernstein, M., Millard, D., eds.: Proceedings of the Hypertext Conference, ACM (2010) in print.
10. Haase, P., Herzig, D.M., Musen, M., Tran, D.T.: Semantic wiki search. In: 6th Annual European Semantic Web Conference, ESWC2009, Heraklion, Crete, Greece. Volume 5554 of LNCS., Springer Verlag (Juni 2009) 445–460
11. Miller, G.A.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review **63** (1956) 81–97