SemSearch 2008, CEUR Workshop Proceedings, ISSN 1613-0073, online at CEUR-WS.org/Vol-334/

# QuiKey – a Demo

Heiko Haller

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

**Abstract.** QuiKey is a light-weight tool that can act as an interactive command-line for a semantic knowledge base. It focuses on highest interaction-efficiency to browse, query and author graph-based knowledge bases in a step-by-step manner. It combines ideas of simple interaction techniques like auto-completion, command interpreters and faceted browsing and integrates them to a new interaction concept. It is being developed in the Semantic Desktop project nepomuk [1]. Despite its versatility, QuiKey needs very little screen space, which also makes it a candidate for future mobile use.

## 1 Idea

QuiKey is inspired by *quicksilver* [2], a kind of advanced application launcher for the Mac that has gained a lot of popularity due to its versatility and efficiency. With very few keystrokes, quicksilver can open files and applications and trigger a large variety of common actions not only on any files but also on specific information objects: Depending on the plug-ins installed, it can e. g. manage play-lists in iTunes, send files via e-mail or dial a contact's phone number.

In knowledge bases like a semantic desktop, knowledge is typically be modelled in a formal and fine granular way. QuiKey provides a light-weight generic UI for browsing and editing them in such fine-granular ways. It also brings simple ways of constructing structured queries to not-so-technically-advanced users.

## 2 Examples / Interaction

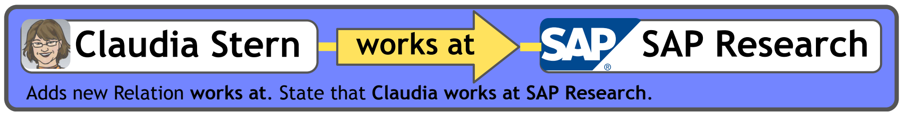

Claudia Stern
works at
SAP Research
Adds new Relation **works at**. State that **Claudia works at SAP Research**.

**Fig. 1.** Mock-up showing how both a new statement and relation are added.

[1] http://nepomuk.semanticdesktop.org/
[2] http://blacktree.com/?quicksilver

QuiKey is organised around the notion of *parts*. A part can be an existing item, a relation, a new text string or a command. Depending on the number, order and types of parts entered, it is decided what action to take.

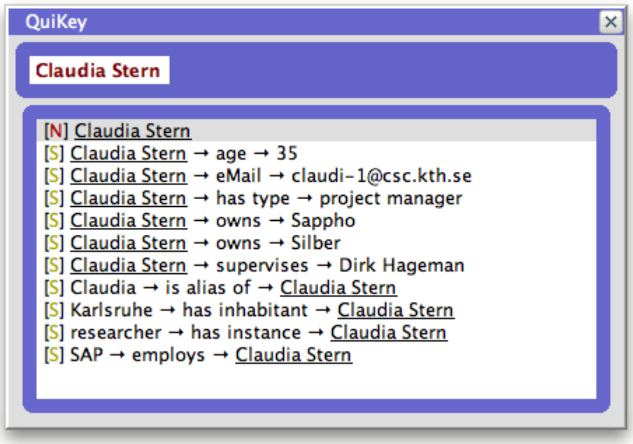
**Authoring** To add a new text item to the knowledge base, it is enough to just type the text into the QuiKey console and press enter. To make statements about existing items, the statement can just be entered in a subject-predicate-object fashion, separated by tab-keys. So. e. g.

```
Claudia Stern→works for→SAP Research[enter]
```

would just add that statement. Only that the user would not even have to type in the whole labels because parts that are already known can be chosen from a list an auto-completion manner with the best fitting NameItem pre-selected. So for this example it is actually enough to type in

```
Ster→wor→SAP R[enter]
```

If not all three parts in such a statement are known strings, the respective items or relations are automatically added to the knowledge base– c. f. Fig. 1. Like this, a knowledge graph can be woven in single simple steps in an ad-hoc fashion. Apart from requiring the user to think in triple patterns, cognitive overhead is reduced to a minimum since additional actions and decisions that are not part of the actual content, like starting an application, opening a new document, finding the right place to add or change content, choosing a file name and location, are not necessary anymore.



Claudia Stern

[N] Claudia Stern
[S] Claudia Stern → age → 35
[S] Claudia Stern → eMail → claudi-1@csc.kth.se
[S] Claudia Stern → has type → project manager
[S] Claudia Stern → owns → Sappho
[S] Claudia Stern → owns → Silber
[S] Claudia Stern → supervises → Dirk Hageman
[S] Claudia → is alias of → Claudia Stern
[S] Karlsruhe → has inhabitant → Claudia Stern
[S] researcher → has instance → Claudia Stern
[S] SAP → employs → Claudia Stern

**Fig. 2.** Screen shot of the current QuiKey implementation showing a list of statements about “Claudia Stern”.

**Browsing** Simply navigating the knowledge base through its graph structure is done with QuiKey without even changing into a different mode: when a part has been selected, before the user types anything new to select the next part, existing contents that fit the part pattern are already displayed in the suggestion area and can be browsed in a way similar to faceted browsing (s. Fig. 2).

**Queries** Constructing complex, possibly nested queries is difficult for non-expert users and every slight error in the syntax makes the whole query fail or return unintended results. QuiKey tackles two common problems:

a) *Misspellings and syntax errors* are largely avoided because instead of requiring the user to write a whole query in some complicated syntax which is parsed later on, in QuiKey the query is constructed interactively, selecting from existing items and without the need of syntactical characters.

b) 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. Simple query items can be constructed with the easy pattern shown in the two examples in Fig. 3:

```
Dirk→knows→?DirksFriends[enter]
```

creates a new query item that represents a query about everyone that 'Dirk' 'knows'.

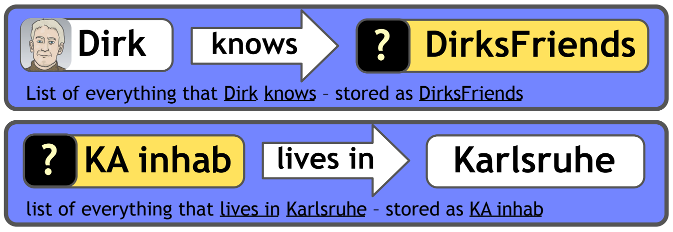

Dirk
knows
?
DirksFriends
List of everything that Dirk knows - stored as DirksFriends
?
KA inhab
lives in
Karlsruhe
list of everything that lives in Karlsruhe - stored as KA inhab

**Fig. 3.** Mock-up of simple elementary queries including generic descriptions of their meaning.

Chained queries like "Who works on a project funded by the EU?" can be asked as shown in Fig. 4. Note that a node or variable between *works on* and *is funded by*, like it is necessary e. g. in SPARQL, can be omitted here since the meaning is clear from the pattern of two relation names after each other. Furthermore, it is consistent with reaching the same query by browsing:

```
EU→funds→
```

would result in a list of everything funded by the EU. Continuing this pattern with

```
EU→funds→has member→
```

would result in a query of all members of these things funded by the EU. Like this browsing and constructing queries becomes the same.

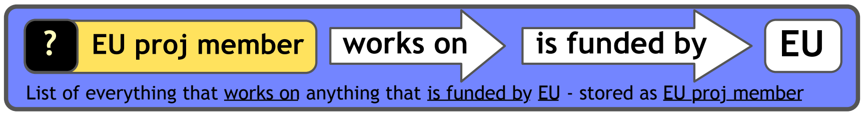

?
EU proj member
works on
is funded by
EU
List of everything that works on anything that is funded by EU - stored as EU proj member

**Fig. 4.** Mock-up of a chained query including generic description of its meaning.

More complex queries can be constructed by combining existing query items like the examples in Fig. 5.

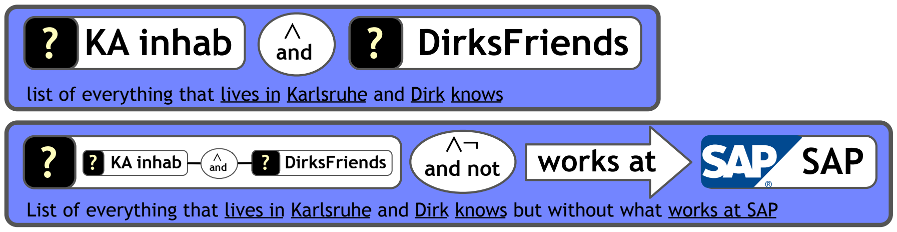

? KA inhab
∧ and
? DirksFriends
list of everything that lives in Karlsruhe and Dirk knows
? KA inhab
and
? DirksFriends
∧¬ and not
works at
SAP
SAP
List of everything that lives in Karlsruhe and Dirk knows but without what works at SAP

**Fig. 5.** Mock-up modular queries combining previously existing query items including generic descriptions of their meaning.

## 3 Technical Background

The current implementation of QuiKey is built on top of CDS ("Conceptual Data Structures"), a lightweight top-level ontology designed to bridge the gap between unstructured content like informal notes and formal semantics like ontologies. CDS allows the use of vague semantics by subsuming arbitrary specific relation types under more general ones. CDS is described in [1] and [2]. The CDS-*framework* which we use as a back-end is a CDS-API in Java, which is designed to serve as a back-end for semantic personal knowledge management tools. It is described in detail in [3].

In CDS there are four basic kinds of items that can be freely added, edited and queried:

**ContentItems** that can hold html-like content

**NameItems** that are characterised by a unique, typically short string – comparable to e. g. a file name or a wiki page name

**relations** i. e. types of relations that can be stated between items (plus, in CDS every relation type has an inverse assigned)

**statements** in the form of subject–predicate–object or rather item–relation–item(in CDS statements are addressable as first-order citizens)

While the general QuiKey approach could be used with any kind of graph-based knowledge base, the CDS framework is especially suited for QuiKey, since NameItems can be used to easily identify items with a unique string using auto-completion mechanisms. And since every relation has an inverse relation defined, any statement can be made and browsed / queried in both directions.

QuiKey will soon also be integrated into the *visual knowledge workbench* of the nepomuk project, e.g. to open existing items directly in the visual iMapping browser [4] or to 'summon' an existing item into a specific place in a map.

The currently used CDS back-end converts the queries to SPARQL. However, since the expressiveness of QuiKey's queries does not exceed EL++[5], there could also be optimised implementations that scale to large knowledge bases without slowing down user experience.

**Acknowledgments:**

Research reported in this paper has been financed by the EU in the Social Semantic Desktop project NEPOMUK (IST-FP6-027705).

## References

1. Völkel, M., Haller, H.: Conceptual data structures (cds) – towards an ontology for semi-formal articulation of personal knowledge. In: Proc. of the 14th International Conference on Conceptual Structures 2006, Aalborg University - Denmark (2006)
2. Völkel, M., Haller, H., Abecker, A.: Modelling higher-level thought structures - method and tool. In: Proceedings of Workshop on Foundations and Applications of the Social Semantic Desktop. (2007)
3. Völkel, M., Haller, H., Bolinder, W., Davis, B., Edlund, H., Groth, K., Gudjonsdottir, R., Kotelnikov, M., Lannerö, P., Lundquist, S., Sogrin, M., Sundblad, Y., Westerlund, B.: Conceptual data structure tools. Deliverable 1.2, nepomuk consortium (2008)
4. Haller, H.: imapping – a graphical approach to semi-structured knowledge modelling. In Rutledge, L., ed.: Proceedings of the The 3rd International Semantic Web User Interaction Workshop (SWUI2006). (2006) Poster and extended abstract presented at the The 3rd International Semantic Web User Interaction Workshop.
5. 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