Design Principles and Realization Techniques for User Friendly, Interactive, and Scalable Ontology Browsing and Inspection Tools

Michael Wessel and Ralf Möller
{wessel | moeller}@racer-systems.com

Racer Systems GmbH & Co. KG
Blumenau 50, 22089 Hamburg, Germany

Abstract. RacerPorter is the default GUI client of the RacerPro OWL DL / description logic system. This paper presents the design principles of RacerPorter as well as the features of the newest version of RacerPorter. RacerPorter has been revised extensively in order to enable it to work with large ontologies (e.g., OpenCyc).

1 Introduction

RacerPorter is the default GUI client of the RacerPro description logic system (DLS). The metaphorical name RacerPorter was chosen to stress that a "user friendly entrance" shall be provided to an otherwise "faceless" RacerPro server, like a hotel porter. Although quite a number of *ontology browsing and inspection tools* (OBITs) as well as *authoring tools* exist and numerous papers have been written about them [1–3], RacerPorter represents a different approach. We presents the design principles behind RacerPorter as well as RacerPorter.

Can such a paper be "scientific"? We think the answer is "yes" if we focus on the more general *design principles* rather than the description of the GUI itself. We have also learned some interesting lessons regarding *scalability.* These insights and *experiences* are of interest to designers of related tools and imply *directions* for further research in the field.

RacerPorter was first released with RacerPro 1.8.1 in 2005. It is used regularly by many RacerPro users. Based on usability feedback and on our own experience with RacerPorter, we have redesigned RacerPorter extensively.[1] Revision was not only required in order to enhance usability, but also in order to solve certain "scalability problems". Users "unscrupulously" load rather large OWL files into RacerPro and expect their taxonomies to be visualized with RacerPorter. Various problems concerning the visualization and navigation were reported. We reacted to these complaints by enhancing the performance and usability of RacerPorter on large KBs. We focused on the OWL version of OpenCyc, see `http://www.opencyc.org/`, which we call `cyc.owl` in the following.

[1] A current beta can be found under `http://www.racer-systems.com/porter`

The RacerPro DLS [4] (visit `http://www.racer-systems.com`) implements the expressive description logic (DL) $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$. As a DLS, RacerPro's most prominent features are the support for so-called *ABoxes* as well as for expressive *Concrete Domains*. Furthermore, it contains a high-performance query and rule language called nRQL. KBs are transmitted using file IO, or over network sockets. RacerPro speaks its own native language, which is an extended KRSS [5] dialect, as well as OWL DL.[2] RacerPro has two network sockets: one for native commands, and a DIG 1.1 [6] port which is used by tools such as Protégé [3]. RacerPro can also read in OWL files directly using its own OWL parser. This will bypass certain difficulties caused by the limitations of the current DIG version 1.1 (especially regarding OWL datatype properties). The specific features of RacerPro (e.g., nRQL) can only be accessed via its native KRSS dialect.

RacerPorter exclusively uses the KRSS port and is thus primarily a *KRSS OBIT*, although special support for OWL is included as well. Compared with DIG, KRSS has the advantage that it can also be used as a *shell language* (DIG was designed under a different perspective). The XML messages standardized by DIG are not on the correct level of abstraction for a shell language (even if a non-XML serialization of DIG messages were used).

In a nutshell, RacerPorter has been designed to meet the following 9 design characteristics:

1. It offers a *KRSS shell for interactive communication with* RacerPro. Already RICE (visit `http://www.ronaldcornet.nl/rice/`) offered a shell.

2. Obviously, *ontology visualization* is important as well. Ontologies have different aspects, i.e., intensional and extensional ones. One can expect that an OBIT is able to visualize taxonomies, role hierarchies as well as ABoxes as graphs and/or trees (of certain kind, using certain graph layout algorithms). An OBIT should thus provides appropriate visualization facilities.

3. Given the fact that OWL KBs tend to become bigger and bigger, appropriate navigation, browsing and focusing mechanisms must be provided, since otherwise the user gets "lost in ontology space". An OBIT must thus provides appropriate (syntactic and semantic) mechanisms.

4. Given that large ontologies such as `cyc.owl` exist, OBITs (as well as the corresponding DLSs, of course) should be able to process and display these large ontologies (scalability aspect).

5. Given that both shell-, gadget- as well as graph-based interactions are offered, the question arises: How to link these interactions, and the results produced by them? An OBIT must provide appropriate solutions.

6. An OBIT should be designed to work *non-blocking* (*asynchronously*). This is especially valuable if large ontologies are processed, and processing takes some time.

7. It is desirable if an OBIT can maintain *different connections simultaneously* to different DLS servers. While on server is busy, the user can change the active connection and continue work with another server.

[2] With the exception of nominals, which are currently only approximated.

8. An OBIT should *avoid opaqueness.* Especially if *modes* are used (and the interface is stateful), then it is necessary to appropriately visualize these modi.

9. Functionality for starting, stopping and controlling DLS servers is desirable. Since each DLS has its proprietary functions and peculiarities it becomes clear that at least part of the OBIT functionality must be tailored for the target DLS.

This paper is structured as follows. We first describe design principles for OBITs and discuss how and why the 9 issues are addressed. Next we describe RacerPorter in more detail and show how the 9 issues are realized. Finally, we discuss some important "scalability" issues and lessons we have learned before we conclude.

2 Design Principles for User-Friendly OBITs

A KRSS or OWL ontology represented in a DLS has many *different aspects: the taxonomy* represents the subsumption relationships between concept names or OWL classes, the *role hierarchy* represents the subsumption relationships between roles or OWL properties, and the ABox represents information about the individuals and their interrelationships (the extensional knowledge). Additional aspects may be present, e.g. queries and rules. Thus, we can make a "shopping list" of "things" which must be accessed, managed and visualized with a DLS OBIT: different DLS servers[3], TBoxes, ABoxes, concepts, roles, individuals, queries and rules, ABox assertions, etc.

In order to avoid an overloaded GUI – which would try to represent these different aspects and aspect-specific functionality in a single window pane – we favor *tabbed interfaces* in order to achieve a clean separation of different aspects. *Different tabs* thus present different aspects of the ontology together with *aspect-specific commands.* The term *different perspectives* also describes the approach quite well.

Whereas many operations are directly performed on the displayed representations of the objects on the RacerPro servers by means of mouse gestures, sometimes called *direct manipulation*, we also *favor push buttons* to invoke commands. In many cases, push buttons will directly invoke KRSS commands, e.g., send an `abox-consistent?` to the connected DLS server. Push buttons also have the neat effect to inform the user directly about commands which are *reasonable to apply* or which can be applied at all in a given situation, simply by being visible, so there is no need to search for a command in a pull-down menu, which distracts focus.

In many cases some *input arguments* must be provided to KRSS commands. Input arguments are provided directly by the user if direct manipulation is employed for the interaction, but with simple push buttons this is not directly possible. Either the user must be prompted for arguments, or a notion of *current objects* must be employed. These current objects have been selected by the user before and are from then on automatically supplied as input arguments to

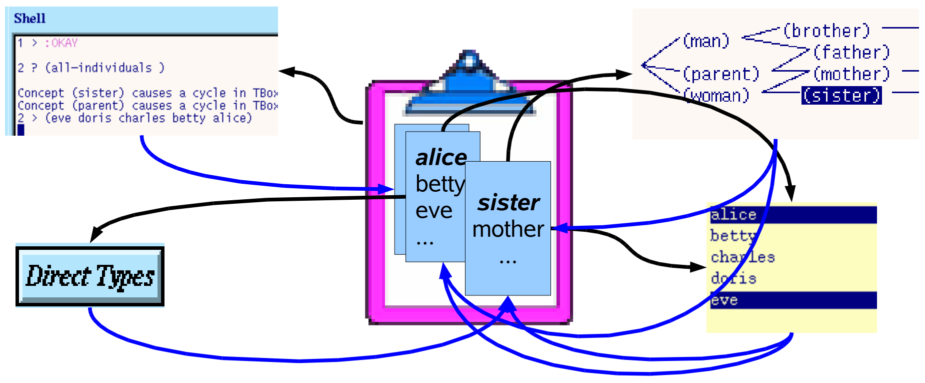
[3] The connection and server settings can be managed using the so-called *connection profiles* which are familiar from networking tools such as SFTP browsers.

KRSS functions. This results in a *stateful GUI.* Sometimes, stateful GUIs are considered harmful. However, we will see that states are unavoidable if non-trivial ontology-inspection tasks shall be performed. Additionally, since also a DLS has a state, this state should be adequately reflected by the GUI as well (which automatically makes it stateful). In order to avoid *opaqueness* it is very important that the current state is appropriately visualized, e.g., in a *status display* which is visible at any time. According to the shopping list, we must thus have a notion of a current DLS server, a current TBox and ABox, current concept, individual and role, current query, etc. These current objects partially constitute the current state of the OBIT.

In many cases, the different tabs of the OBIT visualize different objects. For example, one tab shows the individuals in the *current ABox* (the *individuals tab*), and another tab shows the concepts in the *current TBox* (the *concepts tab*). The information displayed in a certain tab thus depends on the current state. Additionally, the current concept (resp. the current individual) will be highlighted in the concepts tab (resp. the individuals tab), so it can be recognized easily. Two different tabs can also present *the same objects,* but use *different visualizations.* For example, the *taxonomy tab* also presents the concepts in the current TBox, but displays them as nodes in a graph whose edges represent subsumption relationships. Since all tabs display information according to the current state, *the shown information is interrelated.*

However, the interrelatedness of the displayed information achieved so far is not sufficient. For certain ontology inspection tasks, it is also necessary to *establish a kind of information flow* between different tabs. Let us illustrate this need for an input/output flow of information with an example. As described, the *individuals tab* presents the list of individuals in the current ABox. If an individual is selected from that list, it automatically becomes the current individual. In order to explore of which concepts this individual is a direct instance of, we can push the *Direct Types* button shown on the individuals tab which sends the `direct-types` KRSS command to the DLS, using the current ABox and current individual as input arguments. The DLS returns a set of concepts – the result of `direct-types`. In many cases, *further operations* have to be performed on the result just returned, e.g., we might want to learn which *other instances* of these concepts are present in the KB. Thus, the *concepts tab* should provide a functionality which allows to refer to and highlight the just returned concepts, so that subsequent operations can be applied easily on them. An *input/output flow of information between different tabs* is therefore needed.

In order to establish that kind of information flow, we augment the notion of the current state by also including *sets of selected objects* in the state. Thus, the concepts returned by the `direct-types` command can become *selected concepts* (either automatically or with the push of a button). Selected concepts are shown as *highlighted, selected items* in the tabs which present concepts. Moreover, there are also *selected individuals* and *selected roles.* Objects can either be selected manually by means of mouse gestures, or by means of KRSS commands, no matter how they are invoked. All that matters is the notion of selected objects.



```
Shell
1 > :OKAY

2 ? (all-individuals )

Concept (sister) causes a cycle in TBox
Concept (parent) causes a cycle in TBox
2 > (eve doris charles betty alice)
```

Fig. 1. The Clipboard Metaphor

The set of selected objects is also called the *clipboard*; the clipboard metaphor is illustrated in Fig. 1. The selected objects always include the current objects (which are thus "special" selected objects). We also must take into account that the clipboard is *session specific* if the OBIT shall be able to maintain several DLS connections (and thus associated DLS sessions) simultaneously.

As said earlier, a *shell tab* shall be provided for interactive textual communication with the DLS. We claim that only shell-based interactions can offer the required flexibility and expressivity needed for advanced ontology inspection tasks; especially, ad-hoc queries are very important. The shell must be incorporated into this information flow as well. If the `direct-types` command is entered into the shell, then it must be possible to refer to the current ABox as well as to the current individual as input arguments (without having to type or use "copy & paste"). Perhaps the current individual was selected with the mouse in the individuals tab in advance. Again, it must then be possible to *select* the objects returned by the KRSS shell command. Thus, it must be possible to fill the clipboard with results produced by shell commands. This can either happen automatically or by hand, with the push of a button.

Focus control and navigation are two other important issues. It is well-known that the notion of current and selected objects can be used to control the focus. For example, the current concept can provide the root node in the taxonomy graph display. Only the descendants of the current concept will be shown. To browse larger taxonomies, a "depth limit" cutoff on the paths to display can be specified, and an interactive "drill down"-like browsing can be realized if the select node gesture (e.g., a left mouse click) automatically changes the current concept and thus the graph root. If the graph is redrawn immediately, this allows to drill down a large taxonomy interactively and dynamically. However, this automatic graph recomputation *changes the focus*.

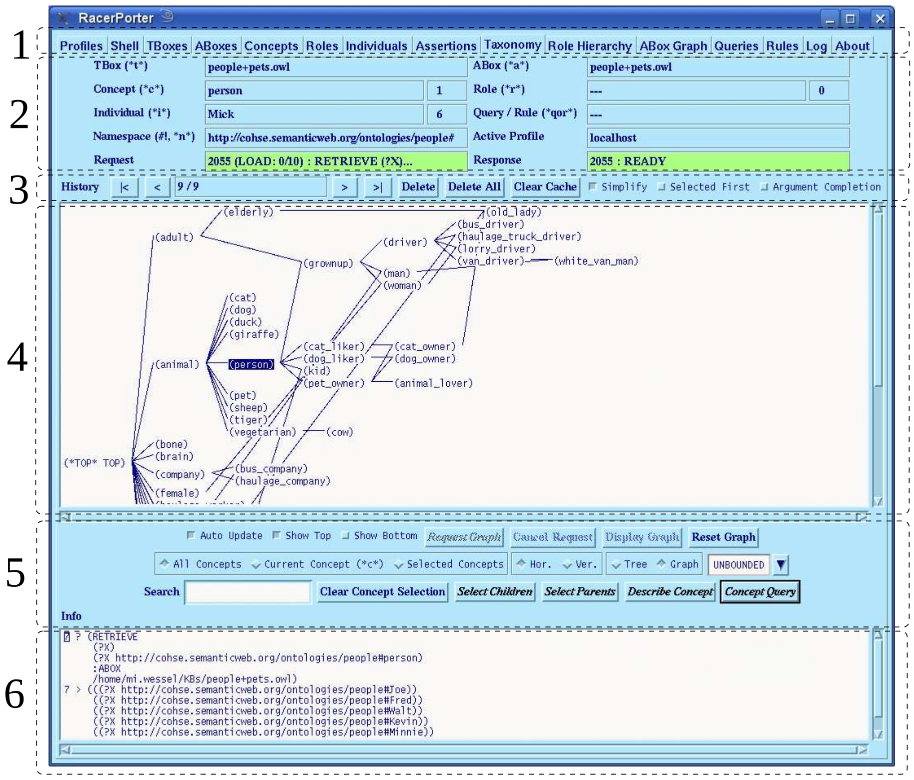
In principle, *changing the focus automatically* can be very distracting. In web browsers, the navigation buttons (back and forth) are thus of utmost importance; they allow to reestablish the previous focus effortless. Thus, a *focus or history navigator* should be present in an OBIT, as also found in Swoop or GrOWL

[1, 7]. However, many users are unhappy with hyperlink-like focus-destroying operations; in web browser, *tabbed browsing* has been invented to address this problem. Thus, we think that the user should be able to determine *when and how the focus is changed* once a gadget is selected.

Sometimes, it is also desirable to *focus on more than one object,* e.g., for ABox graphs. We can simply use the selected objects for that purpose as well. Considering the ABox graph, each selected individual can specify a graph root, and *unraveling* (as understood in Modal Logics) can be used to establish a *local perspective* from that individual's point of view (so there is one graph for each selected individual). This resolves many *visual cluttering* problems. The clipboard is thus not only a structure that enables flow of information, but can also be used to control the focus. This also implies that the focus control is now highly flexible: Since the clipboard can be filled from results of KRSS commands, even *a semantic focus control* is possible. For example, an ad-hoc nRQL query can be typed into the shell, and, with the push of a button, one can focus on the returned ABox individuals in the ABox graph tab. However, one also often wants to focus on individuals simply by their names. Thus, a kind of *Search Field* is needed. Objects that contain the search string get selected automatically. So, this enables *syntactic focusing.* We have observed that many available tools don't offer adequate mechanisms to achieve that kind of information flow and focus control.

Summing up, we conclude that the *current state* must be a vector `<current objects,selected objects,active tab,display options>`. Each time a *state changing operation is performed* by the user (e.g., the current objects or the clipboard is changed), a so-called *history entry* is automatically created, which is just a copy of the current state vector. History entries are inserted at the end of a queue, the so-called *navigation history.* The *history navigator* offers the standard (VCR-like) navigation buttons. The OBIT always reflects the current state, no matter whether this is the latest one or a *historic* state from the history. A history entry only preserves the state information of the GUI, but not the content of the DLS at that time. Thus, a well-known problem arises here: If a historic state is reactivated, then it may no longer be possible to actually refer to the objects referenced by that state, since they may have already been deleted from the DLS. This problem is well-known to WWW users which keep a browsing history. There is no practical solution to this problem (one cannot preserve "copies" of DLS server states in history entries).

We believe that OBITs should allow for asynchronous usage. While a time-consuming command is processed by the DLS, the GUI shouldn't block; instead, the result should be delivered and displayed asynchronously once available. Although the busy DLS will not accept further commands until the current request had been fulfilled (nowadays, there are no true multi-processing DLS), in the meantime the OBIT should a) display status information in order to inform the user what the DLS is currently doing, and b), if possible, allow the user to do other things, e.g., continue editing a KRSS KB, or connect to and work with a different DLS.



TBox (*t*)	people+pets.owl		ABox (*a*)	people+pets.owl
Concept (*c*)	person	1	Role (*r*)	--- 0
Individual (*i*)	Mick	6	Query / Rule (*qor*)	---
Namespace (#!, *n*)	http://cohse.semanticweb.org/ontologies/people#		Active Profile	localhost
Request	2055 (LOAD: 0/10) : RETRIEVE (?X)...		Response	2055 : READY

History: 9 / 9

```
? ? (RETRIEVE
    (?X)
    (?X http://cohse.semanticweb.org/ontologies/people#person)
    :ABOX
    /home/mi.wessel/KBs/people+pets.owl)
? > (((?X http://cohse.semanticweb.org/ontologies/people#Joe))
    ((?X http://cohse.semanticweb.org/ontologies/people#Fred))
    ((?X http://cohse.semanticweb.org/ontologies/people#Walt))
    ((?X http://cohse.semanticweb.org/ontologies/people#Kevin))
    ((?X http://cohse.semanticweb.org/ontologies/people#Minnie))
```

Fig. 2. Linux Version of RacerPorter – The Taxonomy Tab

3 RacerPorter – An OBIT for RacerPro

RacerPorter was designed according to the just explained design principles. Each tab has a very uniform organization, which makes the GUI consistent and comprehensible. With the exceptions of the log tab and the about tab, each tab has **six areas.** Figure 2 shows the taxonomy tab. Let us describe the six areas.

The **first area** shows the available tabs: Profiles, Shell, TBoxes, ABoxes, Concepts, Roles, Individuals, Assertions, Taxonomy, Role Hierarchy, ABox Graph, Queries, Rules, Log, and About tab. The **second area** is the *status display.* It displays the current objects, the current namespace, the current profile (representing the current server connection), as well as the current communication status. The *clipboard* content is not shown, only the cardinality of the sets of selected objects (in the small number fields). The selected objects are highlighted once an appropriate tab is selected. The **third area** shows the history navigator. The **fourth area** is the tab-specific main area. Tab-specific display options and commands are then presented in the **fifth area.** Finally there is the info display which is the **sixth area**. The info area is similar to the shell; however, it only "echos" the shell interaction (accepts no input). All user-invoked KRSS commands are put into the shell (no matter how they are invoked) and are thus also echoed in the info display. This helps to avoid opaqueness, and as a side effect, the user learns the correct KRSS syntax (since also commands invoked by push buttons are echoed).

The taxonomy and the ABox graph tabs use *graph panes* for the fourth area. With the exception of the shell, log and logo tab, the other tabs use *list panes*. List panes allow single or multiple selections of items; selected items represent the selected objects (clipboard). The last selected item specifies the current object. A search field is always present and allows to select objects by name. Selected items will appear on the top of the list if the *Selected First* checkbox is enabled. Some list panes display additional information on their items in multiple columns; e.g., in case of the TBox pane, not only the TBox name is shown, but also the number of concepts in that TBox; DLS server and profile information is shown in the profiles list in appropriate columns, etc.

The *graph panes* are more complicated to handle since they allow to specify *focus*, *layout* as well as *update* options. In case of the ABox graph pane, once can determine which individuals and which edges shall be displayed. Thus, for both individuals and roles, the focus can be set to the current objects, to the selected objects, or to all objects. Appropriate radio buttons are provided. Additional radio buttons control whether only told role assertions, or also inferred role assertions shall be shown; more buttons allow to specify whether the graph display shall be updated automatically if the focus or layout options changes, or whether the user determines when an update is performed. In the latter case, the user first uses the button *request graph* to acquire the information from RacerPro (phase 1). Once the graph is available, the *display graph* button becomes enabled; if pushed, the graph layout is computed and displayed. Both phases can be canceled (and different focus and layout options selected subsequently) if they should take too long.

The *shell tab* provides automatic command completion (simply press the tab key) as well as argument completion. The potential commands and arguments are presented in a pop-up list. Argument completion is supported by accumulating all results ever returned by KRSS commands. In order to make it easy to reexecute commands, the shell maintains its own *shell history* (not to be mistaken with the navigation history). Since the shell is tailored for KRSS commands in Lisp-syntax, we provide *parenthesis matching*, *convenient multi-line input* (simply press enter to start a fresh line), *pretty printing* and *automatic indentation*. Moreover, users no longer have to use full qualified names for OWL resources. In order to fill the clipboard with result objects returned by shell commands, appropriate buttons are provided (e.g., the *Selected Individuals := Last Result* button).

Finally, let us discuss some miscellaneous features. The *log tab* keeps a communication log which can be inspected at any time in order to learn what the DLS is currently doing. The current communication with RacerPro is also visualized in the request and response status fields; appropriate colors are used to visualize the different *stages* of such a communication (request is send, RacerPro performs request, the result is received over the socket, result is parsed, etc.). RacerPorter includes an Emacs-compatible editor with *buffer evalu-*

ation mechanism. Basic functionality to start and stop RacerPro servers is present; startup- and connection-options can be specified with a *profile editor.* We want to stress that RacerPorter is a *multi-session tool;* thus, not only the current state and history, but also the current shell content, is session-specific.

4 Towards Scalable OBITs – Experiments & Experiences

As mentioned, user complained about the performance of RacerPorter on the OWL version of OpenCyc (v0.7.8b, see `http://www.opencyc.org/`). This motivated us to redesign large parts of the original version of RacerPorter. This KB contains 25568 concepts, 9728 roles and 62469 individuals. Given the fact that RacerPorter had problems, we were curious whether the (impressive) tools Swoop, OntoTrack, and GrOWL, could process this KB without problems. Thus, we performed some experiments. All tests were performed on an Intel Core2Duo T5600 @ 1.83GHz with 2 GB main memory running SuSE OSS 10.2.

SWOOP v2.3 beta 3 [1] (release date: 2006-11-10) was able to load `cyc.owl` in less than a minute, very fast. However, we then tried to display the class tree. Unfortunately, after 23 minutes, an "out of heap space memory error" is signaled, even though 1.5 GB of heap space were assigned to the Java VM. We believe that this behavior could be improved easily if appropriate focus resp. concept pre-selection mechanisms were provided.

Next we tried OntoTrack (V 1.00003, downloaded 2007-02-28). Unfortunately, we had no success loading `cyc.owl`, because "An error occurred during loading". According to the authors, this is caused by some incorrect OWL in `cyc.owl` (e.g., properties and individuals with equal names → OWL Full).

It seems that GrOWL [7] currently achieves the best performance on `cyc.owl`. We first put GrOWL into a state in which it only displays the child concepts of `owl:Thing`.[4] After 18 minutes, the graph appeared, and 750 MB were allocated. But when we tried to expand the children of some graph nodes, we encountered problems with the layout algorithm, since too many children were generated. The used (spring) layout algorithms simply needed too much time to compute a layout; however, we could see that the algorithm was working since it updated the graph display every few seconds. It seems that a different algorithm and probably a "non-dynamic display" is needed for `cyc.owl`.

Next we tried the current development version of RacerPro + RacerPorter. Using `owl-read-file`, `cyc.owl` is loaded in 30 seconds, the taxonomy is computed in less than a minute, and transmission to RacerPorter over the KRSS native socket takes approx. 2 minutes. The visualization of the complete class taxonomy having 25.568 nodes needs approx. 8 minutes. However, it is *viable* that the taxonomy is displayed as a graph, and not as a tree, since a tree display results in a combinatorial explosion in the number of nodes (nodes with multiple parents and thus whole subtrees get duplicated). A cancel button can be used to abort the layout process; one can then try again with different options. Considering memory requirements, RacerPorter needed not more than

[4] It should be noted that GrOWL didn't really display the taxonomy, since only the *told* subsumption relationships were visualized (no reasoning was used).

270 MB. We admit that the layout of the *complete taxonomy* is not very useful (too much cluttering), but RacerPorter is capable to display the complete taxonomy. It is definitely necessary to focus on certain parts of the taxonomy. As explained, either the search field or nRQL TBox queries can be used.

We learned that a lot of effort must be put into an OBIT until it can be used successfully on KBs of the size of `cyc.owl`. Many aspects of the original code had to be reworked thoroughly. This not only concerns the choice of appropriate container data structures "that scale", but also issues like communication over socket streams. For example, in our case it was no longer possible to simply coerce sequences of characters read from the socket connected to RacerPro to strings (although this is a very fast operation), since these strings simply get too big to be represented in the Lisp environment we use. On order to reduce socket communication latency, caches must be used, etc.

Summing up, we have presented design principles for OBITs and showed how they are realized in RacerPorter. We want to encourage developers of related tools to make their tools work better on larger KBs. Although the tools are already very impressive, there is certainly room for enhancements, especially regarding visualization and navigation in large KBs. The most serious deficiencies we found in other tools are: 1. the lack of appropriate focus and update control mechanisms, 2. the lack of facilities that support "cancel and retry (with different options)", 3. the lack of information flow between different "plugins" or "views" (no supporting common clipboard or blackboard).

We admit that also RacerPorter still has problems, e.g., the current version is unable to visualize large ABoxes, since the exploited graph displayer can only visualize directed acyclic graphs (DAGs); thus, unraveling is used. Interesting alternatives for a future version are proposed in [8]. Finally, we would like to thank Kay Hidde, Volker Haarslev and our users for valuable ideas and feedback.

References

1. Kalyanpur, A., Parsia, B., Sirin, E., Grau, B.C., Hendler, J.: Swoop: A Web Ontology Editing Browser. Web Semantics: Science, Services and Agents on the World Wide Web **4** (2005) 144–153
2. Liebig, T., Noppens, O.: OntoTrack: A Semantic Approach for Ontology Authoring. Journal of Web Semantics **3** (2005) 116–131
3. Knublauch, H., Musen, M.A., Rector, A.L.: Editing Description Logic Ontologies with the Protégé OWL Plugin. In: Proc. Int. Workshop on Description Logics. (2004)
4. Haarslev, V., Möller, R.: RACER System Description. In: Int. Joint Conference on Automated Reasoning. (2001)
5. Patel-Schneider, P.F., Swartout, B.: Description-Logic Knowledge Representation System Specification from the KRSS Group of the ARPA Knowledge Sharing Effort. Technical report (1993)
6. Bechhofer, S., Möller, R., Crowther, P.: The DIG Description Logic Interface. In: Proc. Int. Workshop on Description Logics. (2003)
7. Sergey Krivov, Rich Williams, F.V.: GrOWL, Visual Browser and Editor for OWL Ontologies. Journal of Web Semantics (2006)
8. Lu, Q., Haarslev, V.: OntoKBEval: A Support Tool for DL-based Evaluation of OWL Ontologies. In: OWL: Experiences and Directions (OWLED). (2006)