A Wiki as an Extensible RDF Presentation Engine

Axel Rauschmayer and Walter Christian Kammergruber

Axel.Rauschmayer@ifi.lmu.de
Walter.Kammergruber@gmail.com
Institut für Informatik
Ludwig-Maximilians-Universität München

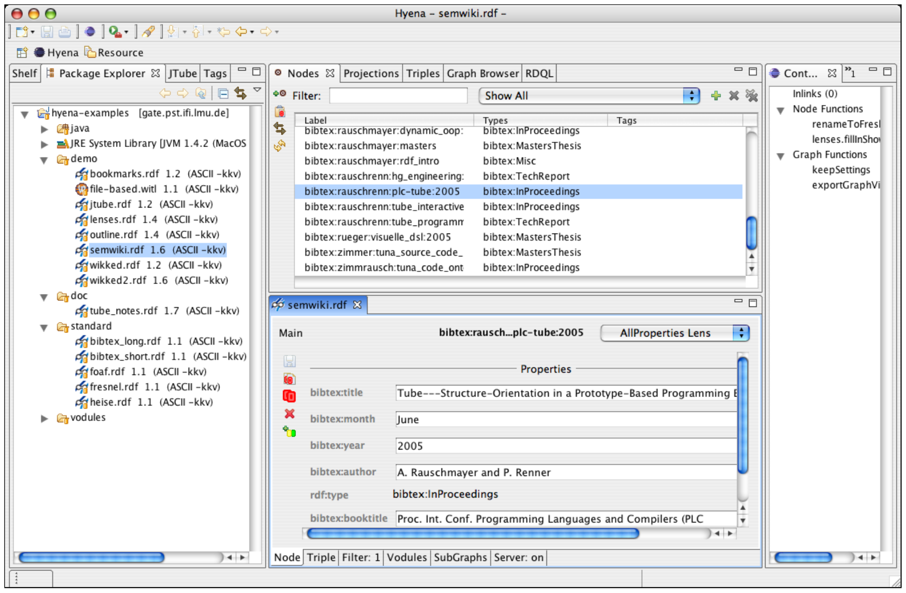
Abstract. *Semantic wikis* [1] establish the role of wikis as integrators of structured and semi-structured data. In this paper, we present Wikked, which is a semantic wiki turned inside out: it is a wiki engine that is embedded in the generic RDF editor Hyena. That is, Hyena edits (structured) RDF and leaves it to Wikked to display (semi-structured) wiki pages stored in RDF nodes. Wiki text has a clearly defined core syntax, while traditional wiki syntax is regarded as syntactic sugar. It is thus easy to convert Wikked pages to various output formats such as HTML and LaTeX. Wikked's built-in functions for presenting RDF data and for invoking Hyena functionality endow it with the ability to define simple custom user interfaces to RDF data.

Keywords: Semantic wiki, RDF, wiki engine, wiki syntax, RDF presentation.

1 Introduction

What would later be called Wikked started as a project about one and a half years ago, when we discussed combining two of our favorite technologies: RDF and wikis. We never thought that this combination would carry us as far as it did (which we think bodes well for the newly-named community of "semantic wikis"): Initially, we had an RDF editor called Hyena and just wanted to mark up and link pages with it. So we implemented the small wiki engine Wikked and embedded it in Hyena. In the end, Wikked grew far beyond its basic functionality and is now even used for constructing small custom user interfaces that display and manipulate RDF. The current focus of Wikked is to complement the GUI-based Hyena and not to replace it with a full-blown web-based RDF editor. One can, however, run Hyena in a web server mode and have Wikked act as a "normal" wiki (Sect. 5).

This paper is directed at members of the semantic wiki community, and thus we neither explain RDF [2, 3] nor wikis [4]. It has the following structure: The next section describes how Wikked fits into the whole of the Hyena picture and what Fresnel lenses are. Sect. 3 explains the core components of Wikked, Sect. 4 walks the reader through an example. Sect. 5 illustrates Wikked's advanced features such as an interactive command line and a web mode. We end the paper by giving related work, future research and a brief conclusion.



```
hyena-examples  [gate.pst.ifi.lmu.de]
  java
  JRE System Library [JVM 1.4.2 (MacOS
  demo
    bookmarks.rdf  1.2  (ASCII -kkv)
    file-based.witl  1.1  (ASCII -kkv)
    jtube.rdf  1.2  (ASCII -kkv)
    lenses.rdf  1.4  (ASCII -kkv)
    outline.rdf  1.4  (ASCII -kkv)
    semwiki.rdf  1.6  (ASCII -kkv)
    wikked.rdf  1.2  (ASCII -kkv)
    wikked2.rdf  1.6  (ASCII -kkv)
  doc
    tube_notes.rdf  1.7  (ASCII -kkv)
  standard
    bibtex_long.rdf  1.1  (ASCII -kkv)
    bibtex_short.rdf  1.1  (ASCII -kkv)
    foaf.rdf  1.1  (ASCII -kkv)
    fresnel.rdf  1.1  (ASCII -kkv)
    heise.rdf  1.1  (ASCII -kkv)
  vodules
```

Label	Types	Tags
bibtex:rauschmayer:dynamic_oop:	bibtex:InProceedings	
bibtex:rauschmayer:masters	bibtex:MastersThesis	
bibtex:rauschmayer:rdf_intro	bibtex:Misc	
bibtex:rauschrenn:hg_engineering:	bibtex:TechReport	
bibtex:rauschrenn:plc-tube:2005	bibtex:InProceedings	
bibtex:rauschrenn:tube_interactive	bibtex:InProceedings	
bibtex:rauschrenn:tube_programm	bibtex:TechReport	
bibtex:rueger:visuelle_dsl:2005	bibtex:MastersThesis	
bibtex:zimmer:tuna_source_code_	bibtex:MastersThesis	
bibtex:zimmrausch:tuna_code_ont	bibtex:InProceedings	

semwiki.rdf

bibtex:rausch...plc-tube:2005

bibtex:title	Tube---Structure-Orientation in a Prototype-Based Programming E
bibtex:month	June
bibtex:year	2005
bibtex:author	A. Rauschmayer and P. Renner
rdf:type	bibtex:InProceedings
bibtex:booktitle	Proc. Int. Conf. Programming Languages and Compilers (PLC

```
Inlinks (0)
Node Functions
    renameToFresl
    lenses.fillInShov
Graph Functions
    keepSettings
    exportGraphVi
```

Fig. 1. A typical Hyena window: To the left is a standard view on the file system. Center bottom is the editor for the currently open file, center top is a view that lists all RDF nodes in that file. The editor shows the content of the currently selected resource. To the right, there is a view displaying the context of the selected resource: incoming links and functions that can be applied to it.

2 Background

2.1 Hyena

Hyena is a generic RDF editor that has been implemented as a plugin for the Eclipse Rich Client Platform [5] Fig. 1 shows what a typical Hyena environment looks like. Hyena's internal architecture is depicted in Fig. 2. Support for an RDF vocabulary in Hyena is provided by plugins called *vodules* (*vo*cabulary mo*dules*). Vodules take care of three different facets of RDF editing: First, a vocabulary has associated declarative data such as OWL constraints, rdfs:comments and rdfs:labels. Accordingly, each vodule can contribute a *subgraph*[1]; every container (or *graph*) of RDF data that is managed by Hyena is composed of several subgraphs. Second, there will always be RDF data formats that cannot be meaningfully edited with a generic editing mechanism. Thus, vodules can implement their own custom widgets for node editing. Third, imperative knowledge might be needed for manipulating RDF data. Vodules add

[1] The reader might know subgraphs under the moniker *named graphs* [6]. We have emulated this feature before implementations in RDF engines (such as Sesame [7] became available, which is why we are still using this legacy name.

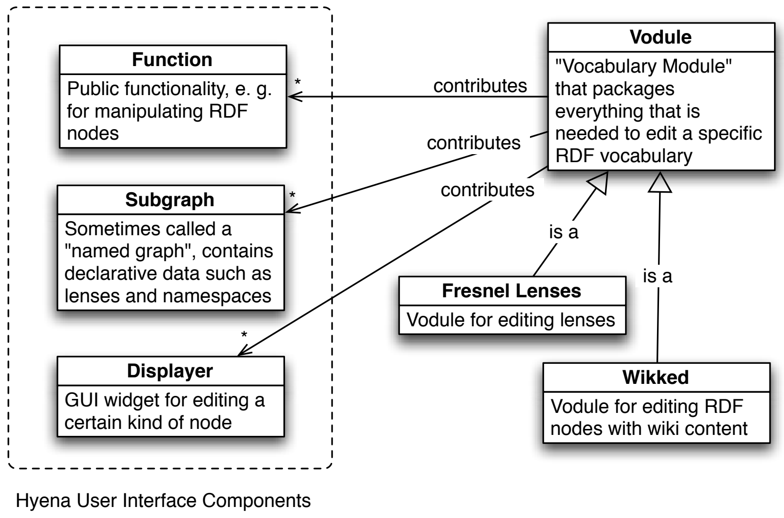

Function
Public functionality, e. g. for manipulating RDF nodes
Subgraph
Sometimes called a "named graph", contains declarative data such as lenses and namespaces
Displayer
GUI widget for editing a certain kind of node
Hyena User Interface Components
Vodule
"Vocabulary Module" that packages everything that is needed to edit a specific RDF vocabulary
contributes
contributes
contributes
*
*
*
is a
is a
Fresnel Lenses
Vodule for editing lenses
Wikked
Vodule for editing RDF nodes with wiki content

Fig. 2. The main components of Hyena's user interface are functions, subgraphs and displayers. A *vodule* supports editing of RDF vocabularies by making cross-cutting contributions to each group of components. Vodules for Fresnel lenses and Wikked pages provide editing support for these vocabularies.

these to Hyena as *functions*. Note that functions can both be accessed via the GUI and as Wikked commands.

2.2 Fresnel Lenses

The *Fresnel Display Vocabulary* [8] is the foundation of Hyena's generic editing facilities. This vocabulary is used to define how nodes of a given type should be displayed. One such definition is called a Fresnel *lens*. The most basic version of a lens needs to state what properties should be displayed and in what order. As an example, here is a minimal lens for the "Friend of a Friend" vocabulary (FOAF, [9]) written in Notation 3 [10] syntax (taken and abridged from [8]):

```
:foafPersonDefaultLens rdf:type fresnel:Lens ;
                       fresnel:classLensDomain foaf:Person ;
                       fresnel:showProperties ( foaf:name
                                                foaf:surname ) .
```

We have given this lens node the URI `:foafPersonDefaultLens` and assigned it the type `fresnel:Lens`. The lens applies to nodes of type `foaf:Person` and displays first and last name (in that order; the value of `fresnel:showProperties` is an `rdf:List`). Advanced Fresnel features include more sophisticated formatting (display a property value as plain text, a link, an image, . . .) and negative definitions (display all properties except X). Fresnel lenses have the advantage that one can deliver data together with instructions on how it is to be displayed. The result of applying a lens to a node is called a *projection* in Hyena.

3 Wikked and WITL (Wikked Template Language)

Wikked is the complete wiki engine, WITL the formal language used for marking up a page. In this section, we will go into details about the design decisions we have taken. They were based on several requirements we had: Initially, We had the RDF editor Hyena and wanted to add notes in rich text. The simplest solution was to start with plain text and support some wiki markup such as headings, lists, tables and text styles. Soon it became obvious that it would be nice to have the ability to display RDF data inside these nodes. For that, we needed a more complete syntax and gave WITL a bracket-based *core syntax* reminiscent of XML. We still support wiki markup by translating it to the core syntax. RDF integration gave us the ability to refer to other nodes. As wiki pages were just another kind of node, we arrived at the most basic wiki functionality: markup and linking. Having a syntax with a proper LL(k) grammar makes it easy to translate our notes to different target markup languages: Apart from the obvious choice of HTML for rendered display of a note, we can also produce LaTeX. This proved handy for situations where one does some quick brainstorming with bullet lists and headings and is able to move on to LaTeX as a more professional publishing format later. The data we presented using WITL was still static. We wanted to add more interactivity and were able to do so by making it easy to extend the WITL vocabulary in Java and by opening up the rest of Hyena to Wikked. Now WITL commands allow one to apply functions to RDF nodes.

The Case for Plain-Text Markup. The most obvious reason we went with plain-text editing is that it is much easier to implement than a true WYSIWYG editor. Hyena's users are very technical, so it was to be assumed that they would be able to handle the challenge. Later, we will still be able to add a WYSIWYG editor. But as WITL is to be frequently extended, typing a plain text is paradoxically often *more* usable than visual editing. LaTeX's ongoing popularity is testament to that. Furthermore, plain text is still the most convenient, most widely used and most cross-platform data format. For example, even if you know nothing about wiki markup, you are still bound to use it if you are sending an email with a bullet list from a Windows computer to a Linux computer and store it in a version control system there. One final consideration is that WITL has to be easier to type than XML, which we otherwise could have used instead. We start this section with a quick example and follow up on it with a more detailed explanation.

3.1 WITL by Example

Before we properly introduce the syntax, we give an example of WITL markup:

```
The {i:core syntax} uses {b:braces} to delimit markup directions.
Simple links follow the same scheme: {http://hypergraphs.de/}.
Further arguments are separated by carets:
```

```
{http://hypergraphs.de^Hypergraphs web site}.

- Then there is also wiki-inspired syntactic sugar
  that is translated into the core syntax.
- Bullet lists, **bold**, ~~italics~~ etc. all work.
```

3.2 Core WITL

At heart, all generic markup languages look very similar: Be it XML, LaTeX or Lisp's S-expressions [11], there are always nested named terms. If you didn't like S-expressions being part of this last list (because they are not really a markup language), you can already guess at the hybrid nature of WITL: On one hand, commands should be easy to type. On the other hand, text will usually dominate, whereas in, say, Java, it can only appear within quotes. So what will our nested terms look like? As they have to be easy to type, XML's named *closing* brackets are out and terms where only the beginning is named are in, as in simple LaTeX commands and S-expressions. Let us start by putting braces around URLs which have to be easy to embed if WIKKED is to be worthy of the "wiki" moniker. We opted for braces, because they appear less frequently in plain text than parentheses and square brackets[2]. We then generalize this syntax to `{tag:argument}` where `http` or `ftp` of an embedded link is just another tag. To allow more than one argument, we introduce the caret "`^`" as an argument separator. Again not for esthetic reasons, but because it appears rarely in plain text (as opposed to the ampersand `&`, less-than `<` or greater-than `>`) and because we will use some of the other candidates (such as the pipe symbol `|`) for wiki markup later. This leads us to expressions such as

```
    Text in {b:{i:bold italics}}
    A link to {a:http://external.com^External Corp.}
```

Comments are written as `{* Comment *}`. Next, if we are to integrate Java, we need a way to express method invocations: creating a list (with the String elements `''a''` and `''b''`) and invoking method `add` on it, to append the element `''c''`, is expressed as `{{List:a^b}add:c}`.

The final two constructs are *raw text* and *pairs*. Raw text is for circumventing WITL when it isn't powerful enough: whereas typical plain text will have critical characters (such as less-than in XML) of the target markup language escaped, raw text is taken verbatim. Therefore one can directly add code in the target language. Raw text syntax is `[[[raw text]]]`. Pairs have two roles: We use them as a basic data structure, among other things to encode maps and we use them for specifying options for a function without getting lost in positional parameters. These two roles are quite related: options are passed to a function as a separate argument containing a map from string to arbitrary data (specified by the user as a sequence of pairs). Pairs are written as `{key=value}`. Fig. 3 gives an overview of the core syntax.

[2] Additionally, we want to keep the option open of using square brackets for citations.

`Plain text`	is default, the following constructs are always started by special character.
`{a:url^text}`	a wikked function, most HTML tags are defined
`{http://foo.com}`	links fit naturally into the syntax scheme
`{{List:xxx}add:12}`	method invocation
`{key=value}`	a pair (e. g. for optional arguments)
`[[[raw]]]`	”raw” or unescaped text; use to insert HTML
`{* Comments *}`	ignored...
`{:varName}`	synonym for `{get:varName}`

Fig. 3. The core syntax of WITL.

3.3 Wiki Markup

While the syntax we have introduced so far is very regular and easy to define, for some of the more commonly used markup, it would be nice to have more “visual” text markup, just like traditional wikis have. Note that for many constructs, this syntax is line-based and largely incompatible with our term/bracket-based core syntax. As we don't want to lose our core syntax, parsing happens in two stages in WIKKED: First, we parse the core terms. Second, we parse lines inside the terms and translate traditional wiki markup to core WITL. While a sequence of lines always starts when a term opens and ends when it closes, a single line can contain both plain text and nested terms. A plain text newline finishes a line. Through this translation step, wiki markup is syntactic sugar for the core syntax. Fig. 4 shows what markup is available. We have intentionally left out breaks and separator lines as these can be cleanly expressed with normal WITL as `{br}` and `{hr}`.

3.4 Java Integration

We initially experimented with turning WITL into a full-blown programming language, but it turned out that that was very cumbersome. We didn't have the development tools we were used to (life without automated refactoring is painful) and WITL syntax is more suited for markup than for programming. As we needed to bridge over to Java anyway[3], we chose to stay very close to Java when it came to defining WITL's semantics and libraries: WITL is a functional language whose values are Java objects. For example, when evaluating the expression `{func:[[[<arg1>]]]^<arg2>}` the first argument is passed to the function `func` as the Java text string `‘‘<arg1>’’`, while the second argument is escaped first and then also passed as a string. Escaping depends on the currently used target language; for HTML the second argument becomes `‘‘&lt;arg2&gt;’’`. Functions cannot be defined (only applied) in pure WITL. They are defined by registering Java objects with the WITL interpreter. Every method in such an object hat has been marked with a special annotation [12] is

[3] It is, after all, the implementation language of WIKKED and HYENA

`==== Heading Level 1 ====`
`=== Heading Level 2 ===`
`== Heading Level 3 ==`
`= Heading Level 4 =`
	Heading Level 1
	Heading Level 2
	Heading Level 3
	Heading Level 4

`Paragraphs`
`are`
 	
`separated by`
`blank lines`
	Paragraphs are
	
	separated by blank lines

`Tabs lead to`
`        indented (quoted) text`
	Tabs lead to
	
	    indented (quoted) text

`**bold**`
`~~italics~~`
`''teletype''`
	bold
	italics
	`teletype`

`- Bullet lists`
`- can be`
`  unordered`
`   + or ordered.`
`   + One can also nest them.`
	• Bullet lists
	• can be unordered
	  1. or ordered.
	  2. One can also nest them.

`: Colons`
`  are for definition lists with`
`: term`
`  and definition`
	Colons
	    are for definition lists with
	term
	    and definition

`|| Table | Heading |`
`| cells | in the |`
`| table | body |`
	Table \/ **Heading**
	cells \/ in the
	table \/ body

`% single-line comment`	

Fig. 4. Wiki markup supported by WITL.

afterwards visible as function in WITL. Before applying a function, WITL compares the number and type of the function arguments to the method signature and makes sure that both correspond. This mechanism facilitates unit-testing and building new functions by composing existing ones. In both cases one can stay in Java and does not have to invoke WITL in any way.

3.5 RDF Integration

A wiki goes from mere text to hypertext by allowing one to link pages. There are two directions of RDF integration in Wikked: First, wikked pages live inside RDF, they are normal RDF nodes, with attached literals that store the page content. We are discouraging the use of blank nodes for wiki pages and automatically generate a URI node when the user wants to create a new page[4]. The reason for this is that blank nodes cannot (stably) be exported as public locations. Interestingly, RDF frees us from purely name-based identification of pages. At the cost of slightly increased complexity, we got rid of problems such as renaming, support for multiple languages and disambiguation.

Second, one can access RDF content with WITL. The most basic functionality here is to link to other nodes which can by either other Wikked pages or arbitrary RDF content. Hyena uses the types of a node to determine whether they should be displayed as wiki pages or as something else. Now, where traditional wikis embed the links to other entities inside the text, this is not a good option for RDF; renaming a node would result in the connection being broken. The solution is to reify that connection in RDF. This happens in two complementary ways:

– Subject linking: Whenever the user has *one specific* RDF node in mind, he just pastes its URI into the WITL source; e. g. as `{subj:http://my-node.com}`. Before saving, Wikked converts the source in the following manner: The page node is used as a `rdf:Seq` of all nodes referenced by subject links. Therefore, the first subject link is the value of property `rdf:_1`, the second one of property `rdf:_2`, etc. The argument of the `subj` function is then changed to be the Seq index. If the above example is the first subject link, it will be changed to `{subj:1}`. All this happens transparently to the user: Before the user edits the page the next time, we put the URIs back in. We have thus moved the actual link out of the source code into a triple. Inserting the link target into the source on-demand guarantees that it is always current. Fig. 5 shows subject linking in use.
– Object linking: If the user has attached *a set of* RDF nodes, he specifies the property he has used, as in `{obj:http://my-predicate.com}`. When displaying the page, Wikked lists all values of that property. Fig. 9 is an example of object linking.

3.6 Skins

Skins are a feature that has been created for the web (or non-embedded) personality of Wikked. When creating web sites, it is often desirable to have reusable

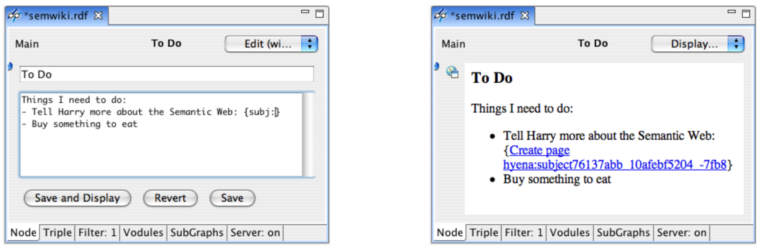
[4] A page node can be easily renamed to a neater URI at any time.

frames for certain subsets of the pages. This is what WIKKED provides with skins: The current display state is denoted as a stack of page IDs (namely, their RDF nodes). The bottom of the stack is the actual content, an arbitrary amount of skins can be pushed on top of it:

Index	Kind of page
...	etc.
2	meta-skin (e. g. content that is the same for the complete web site)
1	skin or meta-page (e. g. a frame for one section of a web site)
0	base page (actual content, without user interface frames)

Rendering of a composite page is performed by going through the stack, starting with the topmost page. Each page is evaluated and passes evaluation on to the next stack element with a special function. The result of the evaluation is usually a text string. This kind of nesting can be compared to around methods in CLOS or super calls in Java. While navigating through the site via links, each transition can specify to either remove or replace elements of the page stack. That is, one can display the same page with a different skin, a different page with the same skin, (prudishly) not show any skin, etc. In case a skin wants to display properties of the content page, we allow it to access the page stack. The index of the content page is always 0, the first skin has index 1, the meta-skin has index 2 and so on. Whenever one starts up WIKKED without specifying what page to display, it fills the stack with a default URI for the start page (index 0) and the start skin (index 1).

4 Example



To Do

```
Things I need to do:
- Tell Harry more about the Semantic Web: {subj:}
- Buy something to eat
```

To Do

Things I need to do:

- Tell Harry more about the Semantic Web: {Create page hyena:subject76137abb_10afebf5204_-7fb8}
- Buy something to eat

(a) Editing the page (b) Displaying the page

Fig. 5. (Left) We pick the title "To Do" for the new page and type the actual content in the text field at the bottom. The empty subject link will get us the option to create a new page. (Right) Using the combo box in the top right corner, we switch the view from WITL source to rendered HTML.

– Let us say we want to use WIKKED to create a page with things we have to do. We use the standard HYENA command for creating a new node and assign it the type `wikked:Page`. Afterwards, we can select a (type-sensitive) displayer for entering the wiki text. We decide on a fitting title for our page and type the text containing the to-do items (Fig. 5(a)). Note that we have not used core WITL, but rather the wiki markup for bullet lists. If we leave the subject argument empty, WIKKED will display a link with which we can create a new page node. Next, we pick a different view of the currently displayed node: whereas up to now, we have looked at WITL source, we now switch to rendered HTML (Fig. 5(b)).

– Before we continue with sketching what we want to tell Harry, we decide that it makes sense to refer him to two papers of ours. This allows us to put WIKKED's RDF integration to use: We parse our BibTeX bibliography with the external BibTeX-to-RDF converter "bibtex2rdf" [13]. Then we use the *Fresnel* display vocabulary [8] to tell HYENA how to display the BibTeX entries. A Fresnel lens declares (per RDF schema class) what properties to display and in what order. HYENA has a built-in lens editor, so creating the lens only involves the following four simple steps: First, create a new lens. Second, add a new property specifying what class this lens applies to. Third, use a predefined HYENA command to collect all used properties from the RDF instances in our RDF graph. Fourth, remove those properties that are to be ignored (none in the example) and rearrange remaining ones via drag and drop until we are satisfied with their order (Fig. 6).

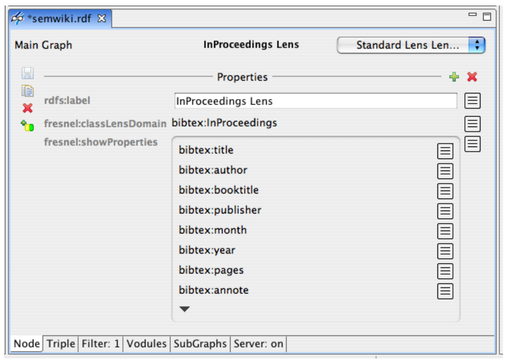
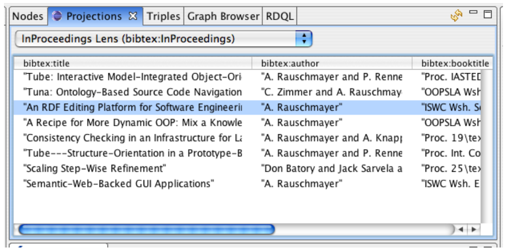

*semwiki.rdf
Main Graph InProceedings Lens Standard Lens Len...
Properties
rdfs:label InProceedings Lens
fresnel:classLensDomain bibtex:InProceedings
fresnel:showProperties
bibtex:title
bibtex:author
bibtex:booktitle
bibtex:publisher
bibtex:month
bibtex:year
bibtex:pages
bibtex:annote
Node | Triple | Filter: 1 | Vodules | SubGraphs | Server: on

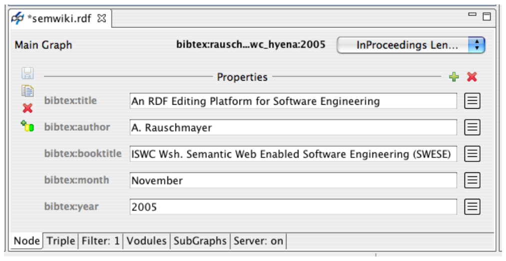
Fig. 6. HYENA has a built-in editor for Fresnel lenses. This figure shows the finished lens where the properties have been automatically filled in by looking at all instances of `bibtex:InProceedings`. The boxes to the right of the predicate names can be dragged and dropped to rearrange the order in which properties will be displayed when using the lens.

– After we have defined the Lens, we want to try it out. First, we use a HYENA view to show us all instances in the current RDF graph that our lens applies to (its *projections*, Fig. 7). Then we click on a list item and have its contents displayed as defined by the lens (Fig. 8).



bibtex:title	bibtex:author	bibtex:booktitle
"Tube: Interactive Model-Integrated Object-Ori	"A. Rauschmayer and P. Renne	"Proc. IASTED
"Tuna: Ontology-Based Source Code Navigation	"C. Zimmer and A. Rauschmay	"OOPSLA Wsh
"An RDF Editing Platform for Software Engineeri	"A. Rauschmayer"	"ISWC Wsh. S
"A Recipe for More Dynamic OOP: Mix a Knowle	"A. Rauschmayer"	"OOPSLA Wsh
"Consistency Checking in an Infrastructure for La	"A. Rauschmayer and A. Knapp	"Proc. 19\tex
"Tube---Structure-Orientation in a Prototype-B	"A. Rauschmayer and P. Renne	"Proc. Int. Co
"Scaling Step-Wise Refinement"	"Don Batory and Jack Sarvela a	"Proc. 25\tex
"Semantic-Web-Backed GUI Applications"	"A. Rauschmayer"	"ISWC Wsh. E

Fig. 7. The "Projections" view in HYENA allows us to list all instances that can be displayed with a certain lens. Here we are displaying all instances of `bibtex:InProceedings`.



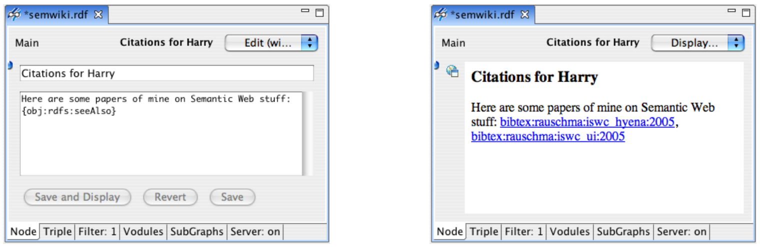
bibtex:title	An RDF Editing Platform for Software Engineering
bibtex:author	A. Rauschmayer
bibtex:booktitle	ISWC Wsh. Semantic Web Enabled Software Engineering (SWESE)
bibtex:month	November
bibtex:year	2005

Fig. 8. One projection of our InProceedings lens. In the top right corner you can see that using the lens is just another way of displaying a node.

– Now we can create the new WIKKED page to hold the citations for Harry, add `rdfs:seeAlso` properties referencing the BibTeX entries, and write a corresponding object link in WITL (Fig. 9(a)). The rendered result is shown in Fig. 9(b).
– To conclude, we do something a little more fancy: we want to have the same set of citations as before, but this time we want them displayed in a bullet list. Furthermore, if someone clicks on a link, we want to search Google for the paper title, in an external browser. This looks as follows:

```
Click on any of these papers to search for them via Google:
{ul:
    {obj:rdfs:seeAlso^^
        {li: {evalLater:{present:{:ANCHOR}}
              ^{util.google:{literal:{:bibtex:title}}}} } } }
```

We use an extended version of the object-linking function `obj` which has three arguments where the first one contains a property URI, the second options and the third an expression that will be re-evaluated once for each object. During each evaluation, the property values *of the current object* will be bound to WITL variables. Within the third argument, function `evalLater` displays a link (whose text is the first argument) and postpones evaluating its second argument until after the link has been clicked. The function we postpone is `util.google` which opens an external browser window with the



Citations for Harry

```
Here are some papers of mine on Semantic Web stuff:
{obj:rdfs:seeAlso}
```

Citations for Harry

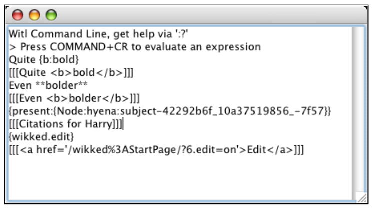
Here are some papers of mine on Semantic Web stuff: bibtex:rauschma:iswc_hyena:2005, bibtex:rauschma:iswc_ui:2005

(a) Editing the page (b) Displaying the page

Fig. 9. (Left) We are using an object link to encode the list of citations we want to tell Harry about. (Right) The citation list rendered via HTML.

result of a Google query. Furthermore, the helper function `present` lets us display a node as "prettily" as possible, i. e. it considers display aids such as an attached `rdfs:label`. Function `literal` extracts the plain text of a literal. Note that we cannot use wiki markup for iteratively creating the bullet list, because the scope of wiki markup does not extend beyond a single function brace.

5 Command Line and Web Serving

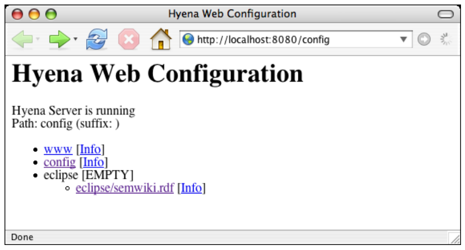


```
Witl Command Line, get help via ':?'
> Press COMMAND+CR to evaluate an expression
Quite {b:bold}
[[[Quite <b>bold</b>]]]
Even **bolder**
[[[Even <b>bolder</b>]]]
{present:{Node:hyena:subject-42292b6f_10a37519856_-7f57}}
[[[Citations for Harry]]]
{wikked.edit}
[[[<a href='/wikked%3AStartPage/?6.edit=on'>Edit</a>]]]
```

Fig. 10. An interactive session with the WITL command line: We evaluate bold markup in two variations; present a prettified version of a wiki page node; and show how editing is added to a page (it can be added to sets of pages via a skin). Note that one can make out in the produced HTML that the start page is currently selected in Hyena. Output is shown as it would be encoded in WITL.

Hyena also comes built-in with a Wikked command line where one can interactively execute WITL code (Fig. 10). What node is currently selected in Hyena is also reflected in the evaluation environment of the command line, so one can explore what source code will look like when evaluated "inside" a page.

Hyena, and thus Wikked, can be started in different modes: It can be started with Eclipse, as a plug-in. But it can also be started stand-alone, obviously with limited functionality. That last mode makes more sense when one



Hyena Web Configuration

Hyena Server is running
Path: config (suffix:)

- www [Info]
- config [Info]
- eclipse [EMPTY]
  - eclipse/semwiki.rdf [Info]

Fig. 11. HYENA's document hierarchy: First is a container `www` that serves static web pages, second is the configuration container `config`, last is an Eclipse container with all currently open documents (currently, only "semwiki.rdf" is open). Every container is accessible via its URI, for example, you are currently seeing the answer to a GET request to the configuration container.

considers that HYENA can also be used for web-serving: either via Jetty, a small embeddable web server or as a servlet[5]. Jetty can be activated in conjunction with Eclipse. As a web server, HYENA manages the currently open files in a named hierarchy of *containers*. A unique name plus the server address results in a component having a unique URI. These URIs are used in HYENA's ReST-based web service [14, 15] API. While each container can provide a customized reaction to web requests the most common use cases are: GET requests display web pages and download data, PUT requests upload RDF files (including wiki definitions), POST requests allow one to react to form input and DELETE requests remove containers. Several standard containers[6] are available: container `www` serves static web pages stored in the standard HYENA directory, container `config` provides configuration and status information about currently active containers (Fig. 11) and, if Eclipse is running, there is an `eclipse` container that publishes all currently open HYENA documents. For example, we can display the running example in the web browser by clicking on `semwiki.rdf`.

6 Related Work

EclipseWiki [16] is a plugin for Eclipse that allows one to edit text files as wiki pages and to embed links to Eclipse resources. In contrast, WIKKED is more specialized: It has RDF integration and can even link to markers *inside* files.

ZML [17] from the Rhizome Wiki has markup rules that are similar to ours, but differs in two important aspects: First, it can "natively" express full HTML which WIKKED foregoes in order to support both LaTeX and HTML; one can still use raw text as a last resort in WIKKED. Second, ZML's syntax is purely line-based where WITL's syntax is a hybrid of bracketed and line-based constructs.

XSDoc Wiki [18] is used for integrating different development artifacts inside a wiki. Instead of RDF, it normalizes all data as XML and has powerful import

[5] Even with Jetty, HYENA establishes connection through a servlet.
[6] More containers can be added programmatically at any time.

facilities. XSDoc has more in common with WIKKED than is apparent at first glance: HYENA partly specializes in using RDF for software engineering. One can, for example, reify Java source code locations as RDF nodes and refer to them in WIKKED. While we do not have XSDoc's flexible format support, we do think that RDF is a better data format when it comes to *linking* and by using Eclipse to track locations, our linking becomes even more robust.

In the Wiki-Templates paper [19], the authors prominently express the need to support structure in wikis. Thus Wiki-Templates provide sophisticated support for structured data entry. Lenses provide much of the same functionality for WIKKED, but projections are not (yet) embeddable inside a wiki page. Furthermore, storing data in RDF has the advantages of being clearly defined and standardized, as opposed to the custom approach taken by Wiki-Templates.

7 Future Research

One way we could go with HYENA is to put more emphasis on its currently underdeveloped web-serving abilities. It would have to be based on Ajax [20] (or Comet [21]) and fully support Fresnel lenses and SPARQL queries. Additionally, we do not currently provide any way of authentication or encryption. Finally, it would be nice to combine blog and wiki into something that has been called a *Bliki* [22]. Because WIKKED is RDF-based and thus highly structured, this would be a very natural extension, as opposed to some of the kludges that currently exist.

8 Conclusion

In this paper, we have shown how HYENA and WIKKED tackle the problem of integrating structured and semi-structured data. This problem is very pertinent to wikis and even more so to semantic wikis. Our answer is separation of concerns: HYENA edits (structured) RDF data, whereas the embedded wiki engine WIKKED is responsible for displaying (semi-structured) wiki content "inside" RDF nodes. We found that this separation of concerns has great usability advantages. It also turns the typical semantic wiki inside out: the wiki is embedded in an RDF editor and not the other way around. WIKKED has been adapted to RDF in one important way: links to data (including other pages) are reified as true RDF relations and not hidden in the wiki text. That means that WIKKED pages are relatively robust regarding changes in the RDF such as node renaming. Finally, WIKKED further enhances HYENA by letting the user invoke HYENA functionality from within a wiki page. As a result, one can make wiki pages more interactive.

Acknowledgements. The authors would like to thank Hubert Baumeister for introducing them to Wikis, when they were still an insider tip; and the SemWiki workshop reviewers for their comments and their contributions to the section on related work.

References

1. Wikipedia: Semantic Wiki. http://en.wikipedia.org/wiki/Semantic_Wiki (2006) [Online; accessed 2006-05-02].
2. Rauschmayer, A.: A Short Introduction to RDF for Software Engineers. (2005)
3. Manola, F., Miller, E.: RDF Primer, W3C Recommendation. http://www.w3.org/TR/rdf-primer/ (2004)
4. Leuf, B., Cunningham, W.: The Wiki Way: Collaboration and Sharing on the Internet. Addison-Wesley (2001)
5. Eclipsepedia: Eclipse Rich Client Platform. (http://wiki.eclipse.org/index.php/Rich_Client_Platform) [Online; accessed 2006-05-02].
6. Carroll, J., et al.: Named Graphs. (http://www.w3.org/2004/03/trix/) W3C Interest Group.
7. Broekstra, J., Kampman, A., Mika, P., et al.: Sesame Home Page. (http://www.openrdf.org/)
8. Semantic Web Interest Group: Fresnel - Display Vocabulary for RDF. http://www.w3.org/2005/04/fresnel-info/ (2005)
9. : The Friend of a Friend (FOAF) project. (http://www.foaf-project.org/) [Online; accessed 2006-05-02].
10. Berners-Lee, T.: Notation3 (N3) a Readable RDF Syntax. (http://www.w3.org/DesignIssues/Notation3.html)
11. Wikipedia: S-expression. http://en.wikipedia.org/wiki/S-expression (2006) [Online; accessed 2006-05-04].
12. Bloch, J., et al.: A Metadata Facility for the Java Programming Language (2002) Java Specification Request 175.
13. Siberski, W.: bibtex2rdf - A Configurable BibTeX to RDF Converter. (http://www.l3s.de/ siberski/bibtex2rdf/)
14. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
15. Prescod, P.: Second Generation Web Services. http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html (2002)
16. Walton, L., Walton, C.: Eclipse Wiki Editor Plugin. (http://eclipsewiki.sourceforge.net/)
17. Liminal Systems: ZML (Zippy Markup Language). (http://www.liminalzone.org/ZML)
18. Aguiar, A., David, G.: WikiWiki Weaving Heterogeneous Software Artifacts. In: Proc. Int. Symp. Wikis, ACM Press (2005)
19. Haake, A., Lukosch, S., Schümmer, T.: Wiki Templates—Adding Structure Support to Wikis on Demand. In: Proc. Int. Symp. Wikis, ACM Press (2005)
20. Garrett, J.J.: Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/essays/archives/000385.php (2005)
21. Russell, A.: Comet: Low Latency Data for the Browser. http://alex.dojotoolkit.org/?p=545 (2006)
22. Fowler, M.: What is a Bliki? http://www.martinfowler.com/bliki/WhatIsaBliki.html (2003)