Fresnel: A Browser-Independent Presentation Vocabulary for RDF

Emmanuel Pietriga[1], Christian Bizer[2], David Karger[3], and Ryan Lee[3,4]

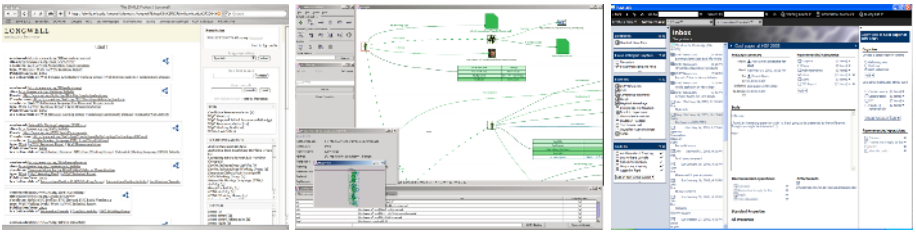
[1] INRIA & Laboratoire de Recherche en Informatique (LRI), Orsay, France
emmanuel.pietriga@inria.fr
[2] Freie Universität Berlin, Germany
chris@bizer.de
[3] MIT CSAIL, Cambridge, MA, USA
karger@mit.edu
[4] W3C (World Wide Web Consortium), Cambridge, MA, USA
ryanlee@w3.org

Abstract. Semantic Web browsers and other tools aimed at displaying RDF data to end users are all concerned with the same problem: presenting content primarily intended for machine consumption in a human-readable way. Their solutions differ but in the end address the same two high-level issues, no matter the underlying representation paradigm: specifying (i) *what* information contained in RDF models should be presented (content selection) and (ii) *how* this information should be presented (content formatting and styling). However, each tool currently relies on its own *ad hoc* mechanisms and vocabulary for specifying RDF presentation knowledge, making it difficult to share and reuse such knowledge across applications. Recognizing the general need for presenting RDF content to users and wanting to promote the exchange of presentation knowledge, we designed Fresnel as a browser-independent vocabulary of core RDF display concepts. In this paper we describe Fresnel's main concepts and present several RDF browsers and visualization tools that have adopted the vocabulary so far.

1 Introduction

RDF (Resource Description Framework) is designed to facilitate machine interpretability of information and does not define a visual presentation model since human readability is not one of its stated goals. Displaying RDF data in a user-friendly manner is a problem addressed by various types of applications using different representation paradigms. Web-based tools such as Longwell [1] (see Figure 1-a) and Piggy-Bank [2] use nested box layouts, or table-like layouts (e.g. Brownsauce [3], Noadster [4], Swoop [5]) for displaying properties of RDF resources with varying levels of details. Other tools like IsaViz [6] (see Figure 1-b) and Welkin [7] represent RDF models as node-link diagrams, explicitly showing their graph structure. A third approach combines these paradigms and extends them with specialized user interface widgets designed for specific information items like calendar data, tree structures, or even DNA sequences, providing advanced navigation tools and other interaction capabilities: Haystack [8] (see Figure 1-c), mSpace [9] and Tabulator [10].

I. Cruz et al. (Eds.): ISWC 2006, LNCS 4273, pp. 158–171, 2006.



(a) (b) (c)

Fig. 1. Various types of RDF browsers: Longwell, IsaViz and Haystack

Such applications are confronted with the same two issues, independently of the underlying representation paradigm and interface capabilities: selecting what content to show and specifying how to format and style this content. Each application takes its own approach and defines its own vocabulary to specify how to present data to users. As with other kinds of knowledge, we believe that being able to share what we consider *presentation knowledge* makes sense in the context of the Semantic Web and that being able to exchange and reuse presentation knowledge between browsers and other visualization tools will benefit both programmers and end users. However, the current diversity of approaches and vocabularies for representing this knowledge makes such exchange and reuse difficult at best, if not impossible.

1.1 Related Work

Early RDF visualization tools rendered RDF models in a predefined, non-customizable way [3]. Recent tools provide more flexible visualizations that can be customized by writing style sheets, transformations, or templates, following either a declarative or a procedural approach.

Procedural approaches consider the presentation process as a series of transformation steps. One such approach consists in using XSLT to transform RDF graphs encoded as RDF/XML trees in an environment such as Cocoon [11]. Authoring XSLT templates and XPath expressions to handle arbitrary RDF/XML is complex, if not impossible, considering the many potential serializations of a given RDF graph and the present lack of a commonly accepted RDF canonicalization in XML [12]. This problem has been partly addressed by Xenon [13], an RDF style sheet ontology that builds on the ideas of XSLT but combines a recursive template mechanism with SPARQL as an RDF-specific selector language. Xenon succeeds in addressing XSLT's RDF canonicalization problem but still has a drawback common to all procedural approaches, that transformation rules are tied to a specific display paradigm and output format, thus preventing the reuse of presentation knowledge across applications.

Declarative approaches are based on formatting and styling rules applied to a generic representation of the content. They can be compared to XHTML+CSS, which has been successful for the classic Web. The Haystack Slide ontology [14], used to describe how Haystack display widgets are laid out, is one example. Another is IsaViz's Graph Style Sheets [15], which modifies the formatting, styling, and visibility of RDF graph

elements represented as node-link diagrams. The main drawback of the declarative approaches developed so far is that they make strong assumptions about, and are thus tied to, the specific display paradigm for which they have been developed and are therefore unlikely to be meaningful across different representation paradigms.

1.2 Toward the Specification of Presentation Knowledge

Providing a single global view of all the information contained in an RDF model is often not useful. The mass of data makes it difficult to extract information relevant to the current task and represents a significant cognitive overload for the user. From an abstract perspective, the first step of the presentation process thus consists in restricting the visualization to small but cohesive parts of the RDF graph, similarly to views in the database world. But identifying what content to show is not sufficient for making a human-friendly presentation from the information. To achieve this goal, the selected content items must be laid out properly and rendered with graphical attributes that favor legibility in order to facilitate general understanding of the displayed information. Relying solely on the content's structure and exploiting knowledge contained in the schema associated with the data is insufficient for producing sophisticated presentations and visualizations. The second step thus consists in formatting and styling selected content items.

Fresnel's goal is to provide an RDF vocabulary to encode information about how to present Semantic Web content to users (i.e., *what* content to show, and *how* to show it) as presentation knowledge that can be exchanged and reused between browsers and other visualization tools. However, we do not expect all applications, which do not necessarily rely on the same representation paradigms and formats, to exchange and reuse all formatting and styling instructions as some might not be appropriate for all paradigms. We therefore identified a set of core presentation concepts that are applicable across applications and which form the core modules of Fresnel. One of the design goals of these modules was to make them easy to learn and use, but also easy to implement in order to promote their adoption by many applications. On top of these modules, we have also begun to define additional Fresnel vocabulary items which are grouped in extension modules. The remainder of this article mainly focuses on the core selection and formatting modules. More information about extension modules can be found in the Fresnel User Manual [16].

2 Core Vocabulary Overview

Fresnel is an RDF vocabulary, described by an OWL ontology [16]. Fresnel presentation knowledge is thus expressed declaratively in RDF and relies on two foundational concepts: *lenses* and *formats* (see Figure 2). Lenses specify which properties of RDF resources are shown and how these properties are ordered while formats indicate how to format content selected by lenses and optionally generate additional static content and hooks in the form of CSS class names that can be used to style the output through external CSS style sheets. The following sections introduce the main vocabulary elements using the examples in Figures 3 and 4.

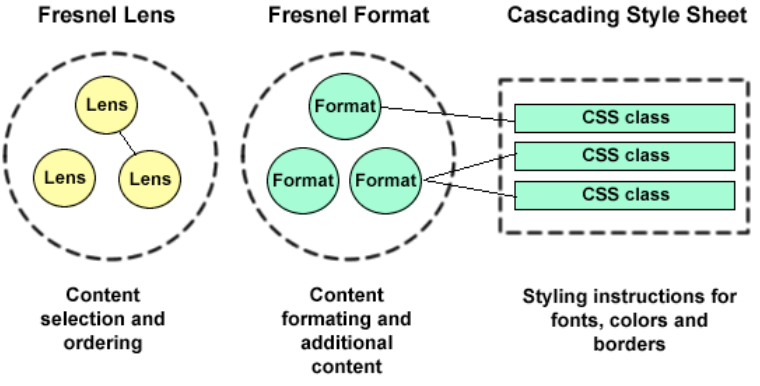


Fig. 2. Fresnel foundational concepts

Figure 3 shows a simple lens and associated formats used to present information about a person described with the FOAF vocabulary [17]. This figure also shows a possible rendering of such a resource, that a browser like Horus [18] or Longwell [1] could produce. Examples use the Notation 3 syntax [19].

2.1 Content Selection

The domain of a lens indicates the set of resources to which a lens applies (line 301: the lens applies to instances of class `foaf:Person`). Property `fresnel:showProperties` is used to specify what properties of these resources to show and in what order (lines 302-308). In this example, the values of both `fresnel:classLensDomain` and `fresnel:showProperties` are basic selectors, which take the form of plain URIs (represented here as qualified names), respectively identifying the class of resources and property types to select. More advanced selection expressions can be written using either FSL or SPARQL. They make it possible to associate lenses with untyped RDF resources, which do occur in real-world models since `rdf:type` properties are not mandatory. They can also be used to specify that a lens should display all properties of a given namespace, or any other complex selection condition(s) that can be represented by an FSL or SPARQL expression (see Section 3).

Fresnel Core provides additional constructs for specifying what properties of resources to display. The special value `fresnel:allProperties` is used when the list of properties that can potentially be associated with resources handled by a lens is unknown to the lens' author but should nevertheless be displayed. When it appears as a member of the list of properties to be shown by a lens, `fresnel:allProperties` designates the set of properties that are not explicitly designated by other property URI references in the list, except for properties that appear in the list of properties to hide (`fresnel:hideProperties`). Two other constructs are used to handle the potential irregularity of RDF data stemming from the fact that different authors might use similar terms coming from different vocabularies to make equivalent statements. Sets of such similar properties can be said to be `fresnel:alternateProperties`. For instance, `foaf:depiction`, `foaf:img` and `p3p:image` could be considered as providing

```
(300) :PersonLens a fresnel:Lens ;
(301)  fresnel:classLensDomain foaf:Person ;
(302)  fresnel:showProperties (
(303)   foaf:name
(304)   foaf:mbox
(305)   [rdf:type fresnel:PropertyDescription;
(306)    fresnel:alternateProperties (
(307)     foaf:depiction foaf:img p3p:image )
(308)    ] ) .

(309) :nameFormat a fresnel:Format ;
(310)      fresnel:propertyFormatDomain foaf:name ;
(311)      fresnel:label "Name" .

(312) :mboxFormat a fresnel:Format ;
(313)      fresnel:propertyFormatDomain foaf:mbox ;
(314)      fresnel:label "Mailbox" ;
(315)      fresnel:value fresnel:externalLink ;
(316)      fresnel:valueFormat [ fresnel:contentAfter "," ] .

(317) :depictFormat a fresnel:Format ;
(318)      fresnel:propertyFormatDomain foaf:depiction ;
(319)      fresnel:label fresnel:none ;
(320)      fresnel:value fresnel:image .
```

Fig. 3. A lens and some formats for presenting instances of class `foaf:Person`

the same information about resources displayed by a given lens. A browser using this lens would try to display the resource's `foaf:depiction`. If the latter did not exist, the browser would then look for `foaf:img` or `p3p:image` (see lines 305-307). Such knowledge can also be represented through ontology mapping mechanisms, but Fresnel provides this alternative as the ontology layer should not be made a requirement of the Fresnel presentation process. The other construct, `fresnel:mergeProperties`, is used to merge the values of related properties (e.g. `foaf:homepage` and `foaf:workHomepage`) into one single set of values that can later be formatted as a whole.

The presentation of property values is not limited to a single level, and (possibly recursive) calls to lenses can be made to display details about the value of a property. Lenses used in this context are referred to as *sublenses*. Modifying the example of Figure 3, we specify in Figure 4 that the browser should render values of the property `foaf:knows` (lines 405-407) using another lens (`PersonLabelLens`, lines 410-413). The FSL expression (see Section 3) on line 406 specifies in an XPath-like manner that only values of `foaf:knows` that are instances of `foaf:Person` should be selected.

The sublens mechanism implies that a lens can recursively call itself as a sublens for displaying property values. In order to prevent infinite loops caused by such recursive calls, Fresnel defines a closure mechanism that allows Fresnel presentation designers to specify the maximum depth of the recursion.

2.2 Content Formatting

The default layout of selected information items is highly dependent on the browser's representation paradigm (e.g. nested box layout, node-link diagrams, etc.), but the final rendering can be customized by associating formatting and styling instructions with elements of the representation.

```
(400) :PersonLens a fresnel:Lens ;
(401)    fresnel:classLensDomain foaf:Person ;
(402)    fresnel:showProperties (
(403)      foaf:name
(404)      foaf:mbox
(405)      [rdf:type fresnel:PropertyDescription ;
(406)       fresnel:property "foaf:knows[foaf:Person]"^^fresnel:fslSelector;
(407)       fresnel:sublens :PersonLabelLens]
(408)    ) ;
(409)    fresnel:group :FOAFmainGroup .

(410)  :PersonLabelLens a fresnel:Lens ;
(411)    fresnel:classLensDomain foaf:Person ;
(412)    fresnel:showProperties ( foaf:name ) ;
(413)    fresnel:group :FOAFsubGroup .

(414)  :nameFormat a fresnel:Format ;
(415)    fresnel:propertyFormatDomain foaf:name ;
(416)    fresnel:label "Name" ;
(417)    fresnel:group :FOAFmainGroup .

(418)  :mboxFormat a fresnel:Format ;
(419)    fresnel:propertyFormatDomain foaf:mbox ;
(420)    fresnel:label "Mailbox" ;
(421)    fresnel:value fresnel:externalLink ;
(422)    fresnel:valueFormat [ fresnel:contentAfter "," ] ;
(423)    fresnel:group :FOAFmainGroup .

(424)  :friendsFormat a fresnel:Format ;
(425)    fresnel:propertyFormatDomain foaf:name ;
(426)    fresnel:label "Friends" ;
(427)    fresnel:group :FOAFsubGroup .

(428)  :FOAFmainGroup a fresnel:Group .
(429)  :FOAFsubGroup a fresnel:Group .
```

Name	Chris Bizer
Mailbox	chris@bizer.de, bizer@gmx.de
Friends	Emmanuel Pietriga Ryan Lee David Karger Stefano Mazzocchi

Fig. 4. An example of a lens using another lens to display some property values

Formats apply to resources, or to properties and their values, depending on the specified domain. The three example formats of Figure 3 apply respectively to the properties `foaf:name`, `foaf:mbox` and `foaf:depiction` (lines 310, 313, 318). Formats can be used to set properties' labels (lines 311, 314, 319). Property `fresnel:label` does not specify a particular layout but simply gives a text string that can be used to identify the property. Labels might already be defined for many properties (e.g., in the associated

vocabulary description using `rdfs:label`), but such labels are not guaranteed to exist. Moreover, a given label might not always be the most appropriate depending on the context in which the property is displayed. For instance, the default label associated with property `foaf:name` in the FOAF schema is *name*. When displaying the persons known by the current person in Figure 4, this default label is replaced by *Friends* (line 426) so as to indicate the appropriate interpretation of the corresponding `foaf:name` property values in this context. The customization of labels also proves useful when displaying property values that are not direct properties of the current resource, as is made possible by the use of SPARQL or FSL expressions such as:

```
foaf:knows/*[airport:iataCode/text() = 'CDG']/foaf:name
```

which would require an explanatory label such as *Friends that leave near Paris*.

Formats can also give instructions regarding how to render values. For instance, line 315 indicates that `foaf:mbox` values should be rendered as clickable links (email addresses). Values of `foaf:depiction` should be fetched from the Web and rendered as bitmap images (line 320).

Property values can be grouped, and additional content such as commas and an ending period can be specified to present multi-valued properties (line 316: inserting a comma in-between each email address). CSS class names can also be associated with the various elements being formatted. These names appear in the output document and can be used to style the output by authoring and referencing CSS style sheets that use rules with the same class names as selectors.

2.3 Lens and Format Grouping

Lenses and formats can be associated through `fresnel:Groups` so that browsers can determine which lenses and formats work together. Fresnel groups are taken into account by browsers when selecting what format(s) to apply to the data selected by a given lens, as several formats might be applicable to the same property values.

Figure 4 illustrates the use of Fresnel groups to display different labels for the `foaf:name` property depending on the context in which the property is shown: the property is labeled *Name* when displayed in the context of the `PersonLens` lens, but is labeled *Friends* when displayed in the context of the `PersonLabelLens` lens. This is achieved by associating the `PersonLens` (lines 400-409) and the `nameFormat` (lines 414-417) to one group: `FOAFmainGroup`, and by associating the `PersonLabelLens` (lines 410-413) and the `friendsFormat` (lines 424-427) to a second group: `FOAFsubGroup`.

A Fresnel group can also serve as a placeholder for formatting instructions that apply to all formats associated with that group, thus making it possible to factorize the declarations. It is also typically used to declare group-wide data, relevant to both lenses and formats, such as namespace prefix bindings.

3 Fresnel Selectors

Selection in Fresnel occurs when specifying the domain of a lens or format and when specifying what properties of a resource a lens should show. Such selection expressions

identify elements of the RDF model to be presented; in other words, specific nodes and arcs in the graph. As we expect selection conditions to be of varying complexity, we allow them to be expressed using different languages in an attempt to balance expressive power against ease of use.

3.1 Basic Selectors

The simplest selectors, called basic selectors, take the form of plain URI references as shown in section 2. Depending on whether they are used as values of `fresnel:instanceLensDomain` or `fresnel:classLensDomain`, these URI references are interpreted respectively either as:

- URI equality constraints (the resource to be selected should be identified by this URI),
- or type constraints (the resources to be selected should be instances of the class identified by this URI).

Basic selectors are also used to identify properties, which are used for instance as values of `fresnel:showProperties` or `fresnel:alternateProperties`.

Basic selectors are easy to use but have very limited expressive power. For instance, they cannot be used to specify that a lens should apply to all instances of class `foaf:Person` that are the subject of at least five `foaf:knows` statements. More powerful languages are required to express such selection constraints.

3.2 Languages for Complex Selection Expressions

Fresnel presentation designers can use two different languages for expressing complex selection expressions. The first option is the SPARQL query language for RDF [20]. In the context of Fresnel, SPARQL queries must always return exactly one result set, meaning that only one variable is allowed in the query's SELECT clause. Figure 5-a gives an example of a lens whose domain is defined by a SPARQL expression. Alternatively, designers who prefer a more XPath-like approach, which proved to be a well-adapted selector language for XSLT, can use the Fresnel Selector Language (FSL). FSL is a language for modeling traversal paths in RDF graphs, designed to address the specific requirements of a selector language for Fresnel. It does not pretend to be a full so-called RDFPath language (contrary to XPR [21], an extension of FSL) but tries to be as simple as possible, both from usability and implementation perspectives. FSL is strongly inspired by XPath [22], reusing many of its concepts and syntactic constructs while adapting them to RDF's graph-based data model. RDF models are considered directed labeled graphs according to RDF Concepts and Abstract Syntax [23]. FSL is therefore fully independent from any serialization. A lens definition using two FSL expressions is shown in Figure 5-b. More information about FSL, including its grammar, data model and semantics is available in the FSL specification [24].

Applications implementing Fresnel are required to support basic selectors, and we expect a reasonable share of them to support the two other languages: SPARQL is gaining

```
# (a) Lens for John Doe's mailboxes    (SPARQL)
:PersonLens a fresnel:Lens ;
            fresnel:instanceLensDomain
              "SELECT ?mbox WHERE ( ?x foaf:name 'John Doe' )
                                  ( ?x foaf:mbox ?mbox )"^^fresnel:sparqlSelector .

# (b) Lens for foaf:Person instances that know at least five other resources  (FSL)
:PersonLens a fresnel:Lens ;
            fresnel:instanceLensDomain
                        "foaf:Person[count(foaf:knows) >= 5]"^^fresnel:fslSelector ;
# and which shows the foaf:name property of all foaf:Person
# instances known by the current resource.
            fresnel:showProperties (
                         "foaf:knows/foaf:Person/foaf:name"^^fresnel:fslSelector) .
```

Fig. 5. Examples of SPARQL and FSL expressions used in Fresnel lens definitions

momentum as a W3C recommendation, and four open-source Java implementations of FSL are already available[1] for HP's Jena Semantic Web Toolkit[2], for IsaViz (providing a visual FSL debugger) and for different versions of the Sesame RDF database[3].

4 Implementations

Fresnel has been designed as an application- and output format-independent RDF presentation vocabulary. In this section we give an overview of various applications implementing Fresnel: Longwell [1] and Horus [18] which both render RDF data as HTML Web pages using nested box layouts, IsaViz [6] which represents RDF graphs as node-link diagrams, and Cardovan, a browser and lens editor based on the SWT GUI toolkit.

Longwell is a Web-based RDF browser whose foundational navigation paradigm is faceted browsing. Faceted browsing displays only the properties that are configured to be 'facets' (i.e., to be important for the user browsing data in one or more specific domains) using values for those fields as a means for zooming into a collection by selecting those items with a particular field-value pair.

The latest version of Longwell relies on the SIMILE Fresnel rendering engine, a Java library built on the Sesame triple store. The engine implements all of the Fresnel core vocabulary and the portion of the extended vocabulary relating to linking groups to CSS stylesheets as well as the option of using FSL as a selector language. The Fresnel engine output consists solely of an XML representation of the Fresnel lenses and formats as they apply to one resource. Longwell then applies an XSLT transformation to the XML to generate XHTML. The default XSLT stylesheet shipped with Longwell will generate a traditional nested box layout, as Horus does, but the stylesheet can be modified by XSLT developers to change the model as they see fit.

The left side of Figure 6 shows the rendering of a `foaf:Organization` resource using a lens that gives some details about the organization and lists its constituent members, all `foaf:Person`s, each listed with their corresponding nickname information to assist in identification.

[1] http://dev.w3.org/cvsweb/java/classes/org/w3c/IsaViz/fresnel/
[2] http://jena.sourceforge.net
[3] http://openrdf.org

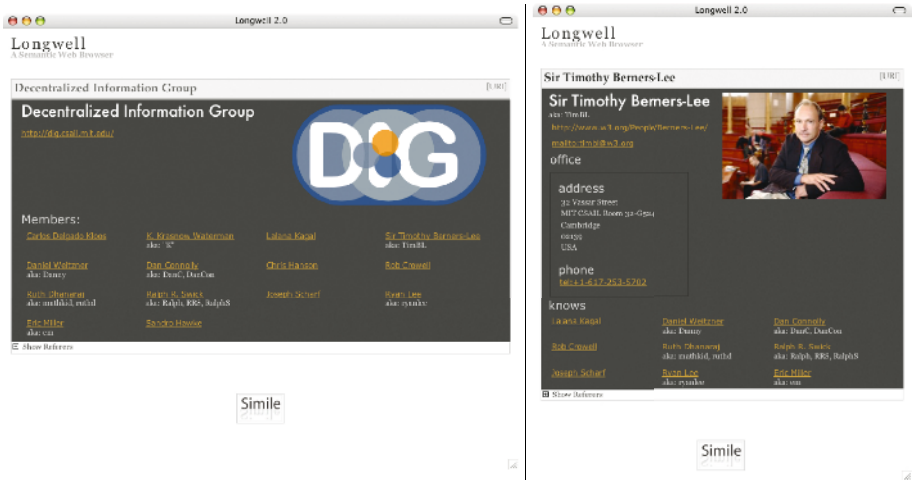


Fig. 6. Displaying a view of an organization (left) and a constituent member (right) in Longwell

The nickname list for each person is preceded by the string 'aka: ', added to the display by using the `fresnel:contentFirst` directive. The list is also comma separated, accomplished by setting `fresnel:contentAfter` to a comma. Clicking on a URI in the display brings the user to that URI; clicking on a textual label changes Longwell's focus to the resource represented by that label.

On the right side of Figure 6, the focus is on one specific member of the organization featured in the left side. A sublens is used to generate office contact details, and the same sublens used in the organization focus (left image) to describe an organization's members is used in the person focus (right image) to describe who this person claims to know.

Horus is an RDF browser that displays RDF information using a nested box layout. The browser provides a simple navigation paradigm for selecting RDF resources and allows users to switch between different lenses for rendering the resources. Horus supports Fresnel lenses and formats, which can be associated together using Fresnel groups. Groups can refer to external CSS style sheets which are used to define fonts, colors and borders. Horus supports basic selectors, but does not offer SPARQL and FSL as selector languages. Horus is implemented using PHP and is backed by a MySQL database. Applying a lens to an RDF resource results in an intermediate tree, which is formatted afterwards using the formats that are associated to the group of the selected lens. The ordered and formatted intermediate tree is then serialized into XHTML.

Figure 7 shows two different views on the same person in Horus. The view on the left uses a lens that displays many details about persons. The sentence "*This person knows the following people*" is a custom label for property `foaf:knows`. The disclaimer "*That a person knows somebody does...*" is static content added using property `fresnel:contentLast`. Some of the links are formatted as external links (`fresnel:value` formatting instruction set to `fresnel:externalLink`), while others refer to RDF resources in the knowledge base, and thus have a different rendering.

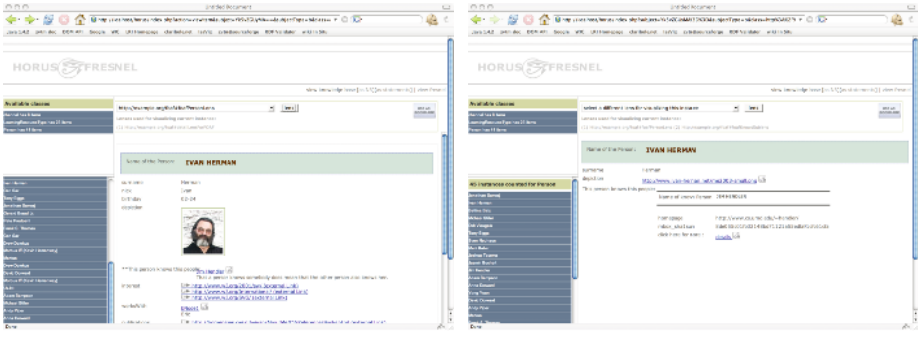


Fig. 7. Two different views on the same person in Horus: detailed view (left), friends view (right)

On the right side of Figure 7, the same person is shown using a different lens. This lens displays less details about the person itself, but refers to a second lens (used as a sublens) for displaying details about other persons known by this person. As the sublens belongs to a different group, another CSS class is used to style the names of the person's friends.

IsaViz is an RDF authoring environment representing RDF models as node-link diagrams. The interpretation of Fresnel in IsaViz is inspired by both Generalized Fisheye Views [25] and Magic Lenses [26]. Fresnel lenses, in conjunction with the formats associated with them through groups, are considered as "genuine" lenses that modify the visual appearance of objects below them.

Figure 8 (left) shows the default rendering of a region of an RDF model containing a `foaf:Person` resource. At this level of magnification, only a few of the many property values associated with the resource are visible. Users need to navigate in the graph in order to get to the values of properties, which can be cumbersome. Alternatively, users can select a Fresnel lens from the list of available lenses loaded in IsaViz through the graphical user interface. The selected lens is then tied to the mouse cursor, and when the lens hovers over a resource that matches its domain, the resource's visual appearance gets modified according to the lens and associated format(s). Resources that match the selected lens' domain are made visually prominent by rendering all other nodes and all arcs using shades of gray with minimum contrast. When the lens hovers over a resource, properties selected by the lens are temporarily rendered with highly-contrasted vivid colors and brought within the current view, closer to the main resource and reordered clockwise according to the ordering of properties in the lens definition, as illustrated in Figure 8 (right). Property values revert back to their original state when the lens moves away from the resource. All these visual modifications, including color and position changes, are smoothly animated thanks to the underlying graphical toolkit's animation capabilities [27], thus keeping the user's cognitive load low following the principles of perceptual continuity.

Fresnel core formatting instructions are interpreted as customizations of the original layout and rendering of nodes and links in the diagram. For instance, nodes representing `foaf:image` property values can be rendered by fetching the actual image from the

Web, as illustrated in Figure 8 (right). The default labels of nodes and arcs can be customized using `fresnel:label` instructions. In case a resource is the subject of multiple statements involving the same property or properties defined as `fresnel:mergeProperties`, the arcs and nodes representing these statements can be merged as a single arc and node with all values within that node, optionally separated by text as specified in `fresnel:contentBefore`, `fresnel:contentAfter` and related formatting instructions.

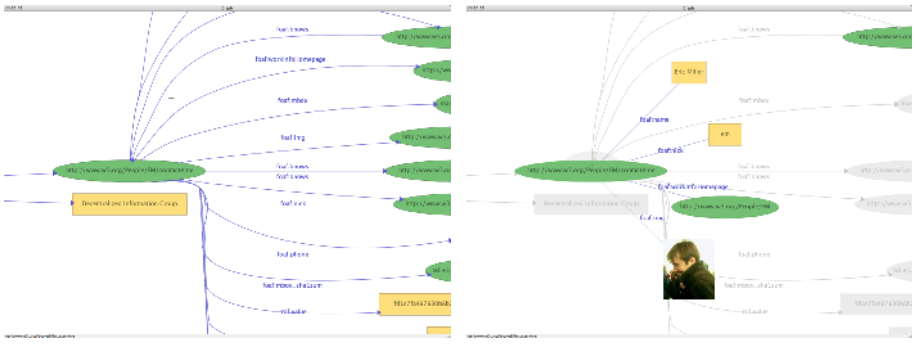


Fig. 8. Zoomed-in view of a `foaf:Person` resource in IsaViz: default presentation (left) and rendered with a Fresnel lens (right)

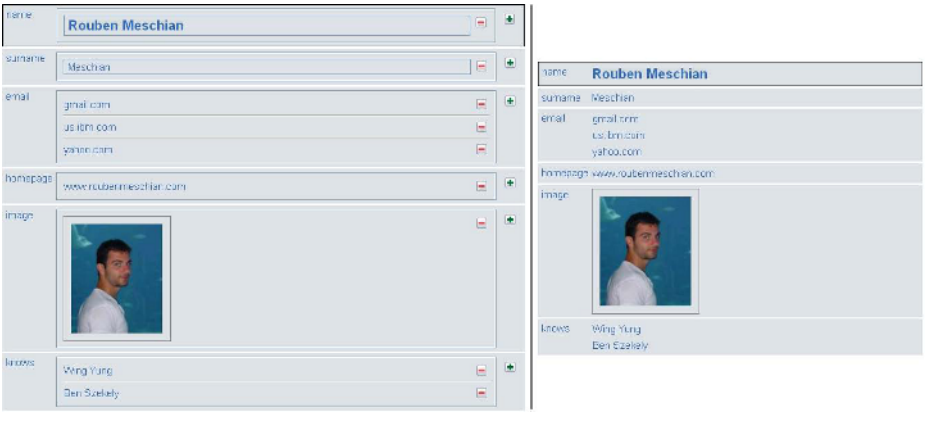


Fig. 9. Editing a lens (left) and visualizing the result (right) in Cardovan

Cardovan is IBM's implementation of Fresnel lenses (see Figure 9). Written in Java, Cardovan renders lenses with the SWT graphical user interface toolkit. Cardovan is similar to other implementations in that it uses a subset of CSS to specify the layout of lens components on the screen. A remarkable feature of Cardovan is that it allows users to modify a lens in place. Users can add new properties to the lens, modify property values, and rearrange the physical layout of the properties displayed, though it is not

a full WYSIWYG Fresnel lens designer. The project is still in its early stages, but is functional and is already being used for internal projects at IBM.

5 Conclusion

We have given an overview of Fresnel, a browser-independent, extensible vocabulary for modeling Semantic Web presentation knowledge. Fresnel has been designed as a modularized, declarative language manipulating selection, formatting, and styling concepts that are applicable across representation paradigms and output formats. We have presented applications implementing Fresnel core modules while based on different representation and navigation paradigms, thus substantiating the claim that Fresnel can be used to model presentation knowledge that is reusable across browsers and other Semantic Web visualization tools.

Although core modules have been frozen for the time being, the Fresnel vocabulary remains a work in progress as new extension modules meeting special needs are being developed (e.g., for describing the *purpose* of lenses and for *editing* information). Extension modules are not necessarily aimed at being application- and paradigm-independent, as they might not be relevant in all cases; but their inclusion in Fresnel provides users with a unified framework for modeling presentation knowledge. Another field for future work is enabling Fresnel formats and lenses to be retrieved transparently from the Web so that RDF browsers could query the Web for display knowledge about previously unknown vocabularies.

The development of Fresnel is an open, community-based effort and new contributors are welcome to participate in it. More information can be found on its Web site `http://www.w3.org/2005/04/fresnel-info/`.

Acknowledgments

We would like to thank Stefano Mazzocchi, Stephen Garland, David Huynh, Karun Bakshi, Hannes Gassert, Jacco van Ossenbruggen, Dennis Quan, Lloyd Rutledge, Rob Gonzalez and Rouben Meschian for their valuable input to the design of the Fresnel vocabulary, their contributions to the discussions on the Fresnel mailing list, and work on Fresnel implementations.

References

1. SIMILE: Longwell RDF Browser (2003-2005) http://simile.mit.edu/longwell/.
2. Huynh, D., Mazzocchi, S., Karger, D.: Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In: Proceedings of the 4th International Semantic Web Conference (ISWC). (2005) 413–430
3. Steer, D.: BrownSauce: An RDF Browser. `http://www.xml.com/pub/a/2003/02/05/brownsauce.html` (2003) XML.com.
4. Rutledge, L., van Ossenbruggen, J., Hardman, L.: Making RDF Presentable: Selection, Structure and Surfability for the Semantic Web. In: Proceedings of the 14th international conference on World Wide Web. (2005) 199–206

5. Kalyanpur, A., Parsia, B., Hendler, J.: A Tool for Working with Web Ontologies. In: Proceedings of Extreme Markup Languages. (2004)
6. Pietriga, E.: IsaViz: A Visual Authoring Tool for RDF. http://www.w3.org/2001/11/IsaViz/ (2001-2006)
7. SIMILE: Welkin. http://simile.mit.edu/welkin/ (2004-2005)
8. Quan, D., Huynh, D., Karger, D.R.: Haystack: A Platform for Authoring End User Semantic Web Applications. In: 2nd International Semantic Web Conference (ISWC). (2003) 738–753
9. mc schraefel, Smith, D., Owens, A., Russell, A., Harris, C.: The evolving mSpace platform: leveraging the Semantic Web on the Trail of the Memex. In: 16th ACM Conference on Hypertext and Hypermedia. (2005) 174–183
10. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: Exploring and Analyzing linked data on the Semantic Web. In: Proceedings of the 3rd Int. Semantic Web User Interaction Workshop, Athens, USA (2006)
11. ASF: The Apache Cocoon Project. http://cocoon.apache.org (2005)
12. Carroll, J.J., Stickler, P.: TriX: RDF triples in XML. In: In the International Journal on Semantic Web and Information Systems, Vol.1, No.1, Jan-Mar 2005. (2005)
13. Quan, D., Karger, D.: Xenon: An RDF Stylesheet Ontology. general@simile.mit.edu mailing list attachment (2004)
14. Huynh, D.: Haystack's User Interface Framework: Tutorial and Reference. http://haystack.lcs.mit.edu/documentation/ui.pdf (2003)
15. Pietriga, E.: Semantic Web Data Visualization with Graph Style Sheets. In: Proceedings of the ACM Symposium on Software Visualization (SoftVis'06), Brighton, UK (2006)
16. Bizer, C., Lee, R., Pietriga, E.: Fresnel - Display Vocabulary for RDF. http://www.w3.org/2005/04/fresnel-info/manual-20050726/ (2005)
17. FOAFers: Friend-of-a-Friend (FOAF). http://www.foaf-project.org/ (2001)
18. Erdmann, T.I., Bizer, C.: Horus RDF Browser. http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/tutorial/horus/ (2005)
19. Berners-Lee, T.: Primer: Getting into RDF & Semantic Web using N3. http://www.w3.org/2000/10/swap/Primer.html (2005)
20. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/ (2005)
21. Cohen-Boulakia, S., Froidevaux, C., Pietriga, E.: Selecting Biological Data Sources and Tools with XPR, a Path Language for RDF. In: Pacific Symposium on Biocomputing (PSB), Maui, Hawaii. (2006) 116–127
22. Clark, J., DeRose, S.: XML Path Language (XPath) version 1.0. http://www.w3.org/TR/xpath (1999)
23. W3C: Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/ (2004)
24. Pietriga, E.: Fresnel Selector Language for RDF. http://www.w3.org/2005/04/fresnel-info/fsl-20050726/ (2005)
25. Furnas, G.W.: A fisheye follow-up: further reflections on focus + context. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, ACM Press (2006) 999–1008
26. Bier, E.A., Stone, M.C., Pier, K., Buxton, W., DeRose, T.D.: Toolglass and magic lenses: the see-through interface. In: SIGGRAPH '93: Proc. of the 20th conference on Computer graphics and interactive techniques, ACM Press (1993) 73–80
27. Pietriga, E.: A Toolkit for Addressing HCI Issues in Visual Language Environments. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), Dallas, USA (2005) 145–152