Fresnel - A Browser-Independent Display Vocabulary for RDF

Christian Bizer[1], Ryan Lee[2], Stefano Mazzocchi[3], and Emmanuel Pietriga[4]

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

Abstract. Todo: Change for new structure of paper: 1. There are different approaches, 2. Abstract Selection and Styling as common problems, 3. Developed Fresnel as a browser-independent ontology to handle these problems in order to faciliate the exchange of display knowledge.
Presenting Semantic Web content in a human-readable way consists in addressing two issues: specifying *what* information contained in an RDF graph should be presented and *how* this information should be presented. Each RDF browser or visualization tool currently relies on its own ad hoc mechanisms and vocabularies for addressing these issues, making it impossible to share RDF presentation knowledge across applications. Recognizing the general need for displaying RDF content and wanting to avoid reinventing the wheel with each new tool, we developed Fresnel as a browser-independent vocabulary of core RDF display concepts applicable across different representation paradigms. Fresnel's two foundational concepts are lenses and styles. Lenses define which properties of an RDF resource, or group of related resources, are displayed and how those properties are ordered. Styles determine how resources and properties are rendered by specifying styling attributes, making use of existing languages such as CSS and SVG wherever possible. In this paper describe the Fresnel display vocabulary and show how Fresnel is used within different RDF browsers, including Longwell and IsaViz.

1 Introduction

Software agents are the primary consumers of Semantic Web content. RDF is thus 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. Todo: Put this in: 1) observe that RDF applications do not always need to do lots of semantic processing (as in the typical RDF-is-for-machines-view), but often only need to show the information in the RDF repository in a human friendly way....However, content encoded in RDF has to be viewed

and understood by humans on many occasions. Displaying RDF in a human-friendly manner is therefore a legitimate concern, addressed by various types of applications using different representation paradigms. Tools like IsaViz [**?**]and Welkin [**?**] represent RDF models as node-link diagrams, where the subjects and objects of RDF triples are the nodes, and predicates the arcs, of the graph. Other tools use nested box layouts (Longwell [**?**]) or table-like layouts (Brownsauce [**?**], Noadster [**?**], Swoop [**?**]) for displaying properties of RDF resources with varying levels of details. A third approach combines these paradigms and extends them with specialized user interface widgets designed for specific information items like DNA sequences, calendar data or tree structures (Haystack [**?**], mSpace[**?**]).

1.1 Selection and Styling

Seen from an abstract perspective generating a visualization is a 5 step process: 1. Select which resources to display. 2. Select which properties to display and how to display property values (DBview/Slice). 3 Select a presentation paradigm 4. Style what has been selected (like CSS). 5. Generate output in the appropriate format.

Step 1 depends on the browsing-paradigm and is not generalizable. Step 3 and 5 depends on the browser and target device and is not generalizable. Step 2 and 4 are generalizable. Thats why we generalize them in order to faciliate the exchange of knowledge.

Todo: Use the text below to make the points above clear.

Providing a single and global view of an RDF model is often not useful. The amount of data makes it difficult to extract information relevant to the current task and is a cognitive overload. The first step thus consists in restricting the visualization to small but cohesive parts of the RDF graph. Users can then select other points of interest by navigating in the model through hyperlinks and refine the selection with paradigms such as faceted browsing [**?**]. Todo refer to DB view or MMS slice[**?**] as similar concepts.

Identifying what content to show is not sufficient to get a human-friendly presentation of the information. To achieve this goal, the selected content items have to 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 not sufficient to produce sophisticated visualizations. The second step thus consists in styling selected content items using explicit styling instructions found in stylesheets.

Relying solely on the content's structure and exploiting knowledge contained in the schema associated with the data is not sufficient to produce sophisticated visualizations. ... That why a way to encode additional display knowledge is needed ...

1.2 Outline of the paper

In section 2 we discuss the various approaches proposed so far for representing display knowledge neccessary for rendering visualizations of RDF graphs and retrieve requirements from them(good and bad paractices). Then we abstract two common problems that are handled in an ad hoc manner by all approaches: Selection and Styling. Based on this abstraction we have developed fresnel as a browser-indpendent...exchange of knowledge.... The Fresnel lens and style vocabularies are described in sections 2 and 3. Section 4 describes the selector languages (having a growing expressiviness) which can be used within Fresnel to identify the elements of RDF graphs lenses and styles should apply to. Section 5 shows how Fresnel can be used within different browsers, including Longwell and IsaViz.

2 Related Work

Todo: Write to fullfil this conclusions/requirements: Declarative better than procedural, layout independent to exchange knowledge, selectors using special language.

There have been various approaches for visualizing RDF data. The approaches can be grouped into three categories: Non-customizable approaches which visualize RDF without taking vocabulary specifiy display knowledge into account[**?**], procedural approaches which encode display knowledge as a series of transformation steps and declarative approaches where display knowledge is represented as a set of generic selection and styling instructions.

2.1 Procedural Approaches

Procedural approaches encode display knowledge as a series of transformation steps. One approach from this category is using XSLT[**?**] to transform RDF/XML[**?**] representations of RDF graphs in an environment like Cocoon[**?**]. 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[**?**]. From a more abstract perspective, working on the XML serialization tree to manipulate RDF graphs, is conceptually wrong since RDF concepts would not be manipulated at the right level of abstraction

The RDF data model is very different from that of XML. Therefore, a language for presenting Semantic Web data should be tailored to the RDF model by taking into account characteristics of the RDF world (potential irregularity of the data, openness and use of different vocabularies to describe resources).

These problems are addressed by Xenon[**?**], an RDF stylesheet ontology that builds on the ideas of XSLT, but combines recursive templating mechanisms with SPARQL as an RDF-specific selector language. Xenon suceeds in addressing XSLT's RDF canonicalization problem but still has the drawback - as all

procedural approaches - that transformation rules are very closely bound to a specific display paradigm or output format preventing the reuse of display knowledge across applications.

2.2 Declarative Approaches

Todo: talk about what is right in CSS: Declarative and seperation of content and style.

Declarative approaches represent display knowledge as a set of generic selection and styling instructions. They try to copy the ideas of HTML+CSS that where sucessfull in the classic web.

Several browsers and visualization tools provide RDF display vocabularies to model presentation knowledge [**?**,**?**,**?**]. Each vocabulary is however closely tied to one of these tools, making it difficult to share and reuse presentation knowledge across applications. Recognizing the general need for modeling and sharing display knowledge, and wanting to avoid reinventing the wheel with each new visualization tool, we developed Fresnel as a browser- and format-independent vocabulary of core RDF display concepts.

Talk about: GSS [**?**] (negativ: node-and-arc oriented, selector language in RDF), Haystack Slides [**?**](negativ: layout oriented, redo of HTML and CSS, HTML font tag problem) , Longwell config (too simple), ... Also talk about [**?**].

3 Selection and Styling with Fresnel

Todo: Discuss all general stuff about fresnel goals and solution in this section. ... Solve problems above in a general and declarative way in order to be browser- and presentation-paradim-independent. Fresnel relies on two foundational concepts: *Lenses* and *Styles* (Figure 1). Lenses define which properties of RDF resources are displayed and how these properties are ordered. Fresnel styles determine how the properties are rendered. They contain RDF-specific styling instructions and hooks to CSS and SVG styling instructions, which are used to specify fonts, colors, margins, borders, and other decorative elements. Todo: How is the styling approach different to HTML,SVG ....

Fresnel adheres to a strict separation between content selection and styling. Fresnel's standard presentation process involves three steps. First, the parts of the RDF graph to be presented are selected and ordered using Fresnel lenses. The result of this selection step is an ordered tree of RDF nodes, not containing any styling information yet. Fresnel styles are then applied to this tree, associating styling instructions to its nodes. The result of this second step is finally rendered as a document in the desired output format. This strict separation of the two steps allows applications to only implement the aspects of Fresnel they are interested in. For instance, visualization tools displaying entire RDF graphs without performing any content filtering can choose to support Fresnel styles but ignore Fresnel lenses. Other tools might choose to use Fresnel lenses for content selection but define their own template mechanism for content presentation. As

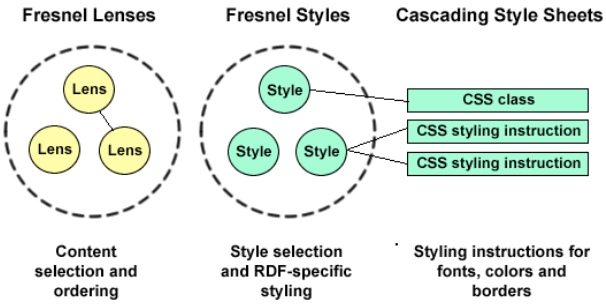

Fig. 1. Fresnel's foundational Concepts

mentioned earlier, Fresnel is further modularized, grouping concepts into core and advanced modules so that applications only interested in basic Fresnel functionalities can support limited but well-identified subsets of the vocabulary.

Fresnel has been designed as an RDF vocabulary in order to facilitate the exchange of presentation knowledge between browsers. The RDF format should also facilitate the discovery and composition of new presentation knowledge originating from different sources; one goal of Fresnel is indeed to allow browsers which are confronted with unknown ontologies to search the Web for presentation knowledge applicable to these ontologies. Moreover, published Semantic Web data tends to be irregular and to mix vocabularies for the description of resources. Experience from the FOAF community shows that different information providers describe resources at different levels of detail. RDF typing information is also often omitted. Fresnel has therefore been designed to handle heterogeneous vocabularies and requires minimal data consistency.

Inspired by the modularization effort in recent Web standards, Fresnel concepts are divided in core and advanced modules so that applications only interested in basic Fresnel functionalities can support limited but well-identified subsets of the vocabulary. Core presentation concepts are abstract enough to be applicable across different representation paradigms and output formats. Browsers that have more advanced modeling requirements can implement the advanced modules but also define their own extensions while remaining able to exchange presentation knowledge using the core concepts and associated terms they share with other visualization tools. Furthermore, Fresnel builds on existing Web technology wherever possible. Styling instructions are mainly drawn from CSS [**?**] and SVG [**?**], and new terms are introduced only for RDF-specific concepts that are not covered by existing Web standards.

4 Fresnel Lens Vocabulary

Fresnel lenses describe which properties of RDF resources are shown and how these properties are ordered. For example, a summary lens for a class represent-

ing people such as `foaf:Person` might display the name and the email address properties of each person.

The `fresnel:lensDomain` property specifies the set of instances to which a lens is applicable. A lens domain can be defined by one or more classes, in which case the lens is applicable to instances of these classes. A domain can also be defined by a set of instances using an FSL or SPARQL selector (see section 7). Such selectors are used to specify a lens domain in terms of the existence (or lack) of properties and values associated with the resources that constitute the domain. They thus make it possible to associate lenses with untyped RDF resources, which can and do occur in real-world models as `rdf:type` properties are not mandatory.

In a distributed and open environment, browsers confronted with unknown RDF vocabularies for which no stylesheet is available locally can query repositories for appropriate lenses. Queries are made on the `fresnel:lensDomain` property of available lenses in order to evaluate their ability to handle the resources to be presented. Queries can also make use of the `fresnel:purpose` property, which encodes metadata about lenses, more specifically their intended use and characteristics. The purpose property can state that a lens is the default lens for a given class, or that it gives a good one-line summary (e.g. a label) of resources, etc. The following example shows a lens applicable to resources identifying persons, as defined by the Friend-of-a-Friend (FOAF) vocabulary [**?**].

```
:PersonLens a fresnel:Lens ;
    fresnel:lensDomain foaf:Person ;
    fresnel:purpose fresnel:defaultLens ;
    fresnel:showProperties (
        foaf:name
        foaf:mbox
        [ a fresnel:PropertyDescription ;
          fresnel:property foaf:knows ;
          fresnel:depth "2"^^xsd:nonNegativeInteger ;
          fresnel:sublens :PersonLens ]
        fresnel:allProperties ) ;
    fresnel:hideProperties (
        rdfs:label
        dc:title ) .
```

4.1 Property Selection and Ordering

In the previous example[5], `fresnel:showProperties` and `fresnel:hideProperties` define which properties must be shown or hidden when the lens is used to display resources of type `foaf:Person`. The value of `showProperty` and `hideProperties` can either be a single URI reference identifying the property to show or hide, or a list of such URI references. Properties to show are displayed according to their order of appearance in the list that is the value of `fresnel:showProperties`.

Special value `fresnel:allProperties` can be used to avoid having to explicitly name each property that should be displayed. This value is also useful when the

[5] All examples in this paper use the Notation 3 syntax for RDF [**?**].

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`). These unnamed properties are displayed according to the position of `fresnel:allProperties` in the list. In the previous example, `foaf:name`, `foaf:mbox` and `foaf:knows` properties are displayed in this order, before all other properties, which appear next as indicated by the presence of `fresnel:allProperties` at the end of the list of properties to be shown. Properties `rdfs:label` and `dc:title` will not be displayed even if they exist, as they are explicitly declared as hidden.

Fresnel provides two additional constructs for specifying what properties of resources to display. The first one handles the potential irregularity of RDF data coming 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 grouped in ordered lists and said to be `fresnel:alternateProperties`. For instance, `foaf:name`, `dc:title`, `rdfs:label` can be considered by a lens as giving the same information about resources. A browser using this lens will then try to display the resource's `foaf:name`. If the latter does not exist, the browser will look for `dc:title` and `rdfs:label` in this order. The second Fresnel 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 for presentation purposes.

4.2 Lenses and Sublenses

It is often desirable to display cohesive parts of RDF graphs involving a group of related resources, such as a person together with her projects and papers. Cohesive parts of graphs are defined by relating lenses using the `fresnel:sublens` property. In the previous example, a sublens is used to display persons that are known by the current person. The `fresnel:sublens` property states that lens `:PersonLens` should be used to display the value of `foaf:knows` properties. As this introduces recursiveness in the lens definition, property `fresnel:depth` is used to specify the closure value. In the end, this example lens will display persons together with her friends and the persons known by her friends.

As shown in this example, lens URIs are used to explicitly identify what lens to use as sublenses for presenting property values. Appropriate sublenses can also be identified by expressing constraints on lens definitions (domain, purpose, etc.) using a Fresnel Selector Expression or a SPARQL query evaluated against the set of lenses available to the browser.

5 Fresnel Style Vocabulary

Fresnel lenses specify what information to display but do not give any indication about how to display it. The representation of selected information items mainly

depends on the browser's representation paradigm (e.g. nested box layout, table layout, node-link diagrams, advanced widget-based UI, etc.) which defines a default rendering method. The final rendering can however be customized by associating specific styling and layout instructions to elements of the representation, as CSS styling rules do for elements of HTML documents.

Fresnel styling rules are made of a selector and a set of associated styling instructions. Following the lens approach, property `fresnel:styleDomain` takes a selector as its value (see section 7) and is used to declare the set of RDF properties or resources to which styling instructions apply. Styling instructions themselves are expressed using either Fresnel constructs for RDF-specific styling concepts, or existing languages (CSS [**?**] and SVG [**?**]) for fonts, colors, margins, and borders.

5.1 Fresnel Styling Instructions

Fresnel styling instructions are expressed in RDF and can be used to specify RDF-specific styling concepts such as how properties are labeled and grouped. Fresnel's default behavior is to label properties using the declared `rdfs:label` of the property type. If this information is not available, then the property's URI is displayed instead. This behavior can be customized thanks to property `fresnel:label`, which defines whether a property label is displayed (`fresnel:show`) or not (`fresnel:none`, see figure 4), or if it has a fixed custom literal value.

The presentation of property values can also be customized. Fresnel's default property value display method consists in searching for a lens matching the value's type and characterized by purpose `fresnel:labelLens`. Some kinds of values are however better represented by other means than labels, and it is therefore desirable to specify alternate behaviors. Instruction `fresnel:value` can be associated with different Fresnel behaviors, such as `fresnel:image` (see figure 4) which states that the value to be represented is a URI reference identifying a bitmap image whose content should be fetched over the Web in order to be displayed. Property values can be grouped, and additional content such as commas and an ending period can be specified using instructions `fresnel:contentAfter` and `fresnel:contentLast` to present multi-valued properties. Instruction `fresnel:contentNoValue` can be used to generate fixed content signaling missing values.

5.2 CSS and SVG Styling Instructions

As the set of display elements and layout method depend entirely on the browser's representation paradigm, Fresnel only defines an abstract representation model that can be interpreted and instantiated differently by every application. This abstract model is used to identify the elements of the output document to which CSS and SVG styling instructions can be hooked in a cross-application and cross-format manner. In the abstract model, the container box is the outermost Fresnel container. It identifies the region of the display that contains a group of principal resources to be presented. The container box contains resource and

property boxes which identify regions of the display that are respectively associated with resources themselves and their properties. Property boxes are further decomposed as a label box and one or more value boxes which respectively hold the property's label and its value(s). Styling instructions can then be hooked to these presentation elements. The instructions are specified as literal values of properties `fresnel:containerStyle`, `resourceStyle`, `propertyStyle`, `labelStyle` and `valueStyle`. Such literal values can contain either styling instructions directly (see figure 4), or a CSS class name referencing a rule in an external stylesheet.

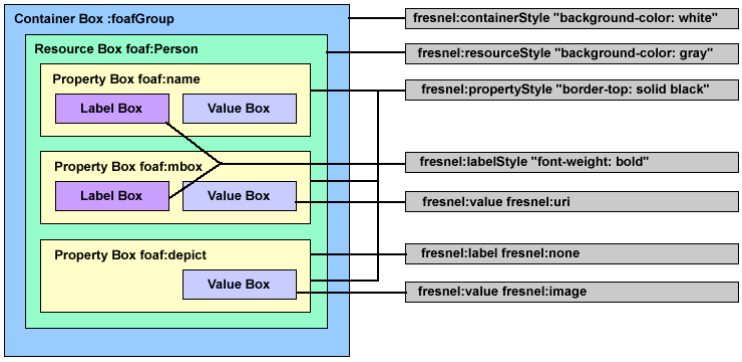

Fig. 2. Box model with attached styling instructions

Figure 2 contains a schematic view of how the abstract Fresnel model could be interpreted and instantiated by a Web-based browser using the nested box representation paradigm.

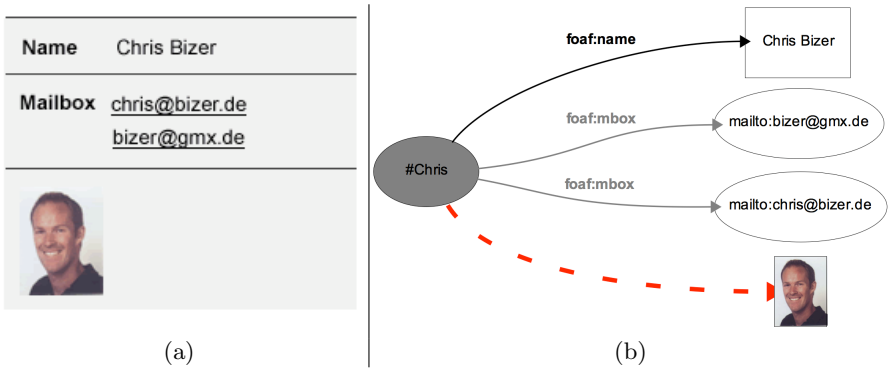

Fig. 3. Two interpretations for two representation paradigms

Figure 3-a shows a possible final rendering based on this model as it might be generated by Longwell. Note that the example model of figure 2 is just one possible instantiation of the abstract Fresnel model. The abstract presentation elements introduced above could be mapped to (possibly non-rectangular) shapes laid out in very different manners, depending on the browser's capabilities and fundamental representation paradigm.

Figure 3-b shows a different final rendering produced by IsaViz[6], based on another instantiation of the Fresnel abstract representation model relying on node-link diagrams. The two examples of figure 3 thus illustrate two presentations, by two browsers based on different representation paradigms, of the same data using the same Fresnel lens and styling instructions. But other representation models based on other paradigms are possible. For instance, a server producing HTML pages for user agents that do not support CSS might choose a basic table-based representation of RDF triples and interpret the property and value boxes as cells of a table row.

The style example of figure 4 specifies that `foaf:depiction` property values should be displayed as images with a black border, but no property label. Additional styling instructions are used to further customize `foaf:depiction` properties' appearance. Some of these instructions might be ignored by browsers depending on their relevance with respect to the underlying representation paradigm. For instance, the border-top instruction will be interpreted by browsers based on a standard box model, whereas it is likely to be ignored by visualization tools such as IsaViz which is more likely to interpret stroke-related instructions.

```
:depictStyle rdf:type fresnel:Style ;
    fresnel:styleDomain foaf:depiction ;
    fresnel:label fresnel:none ;
    fresnel:value fresnel:image ;
    fresnel:propertyStyle "border-top: solid
                           black"^^fresnel:StylingInstructions;
    fresnel:labelStyle "stroke-dasharray: 10px,20px; stroke-width: 2px;
                        stroke: red"^^fresnel:StylingInstructions ;
    fresnel:valueStyle "border: solid black"^^fresnel:StylingInstructions.
```

Fig. 4. Fresnel style for property `foaf:depiction`

6 Fresnel Selectors

Selection in Fresnel occurs at different levels: when specifying what properties should be displayed or hidden (`fresnel:showProperties`, `fresnel:hideProperties`), and when specifying the domain of a lens or style (`fresnel:lensDomain`, `fresnel:styleDomain`). All four properties take as values expressions that identify

[6] This Fresnel presentation was simulated with a GSS stylesheet [**?**] as IsaViz does not support all Fresnel features at the time of submission.

elements of the RDF model to be presented, in other words specific nodes and arcs in the graph. Three languages can be used in Fresnel for specifying selection expressions, offering increasing levels of expressive power and complexity.

6.1 Basic Selectors

The simplest selectors in Fresnel take the form of URI references. Depending on its context of use, a URI reference can identify a resource, a class of resources, or a type of property, as shown in the following examples.

```
:PersonLens a fresnel:Lens ;
    fresnel:lensDomain foaf:Person ;
    fresnel:showProperties ( foaf:name
                             foaf:depiction ).
```

Property `fresnel:lensDomain` indicates that this lens should be used to display instances of class `foaf:Person`, i.e., resources that have an `rdf:type` property pointing at class (or at a subclass of [7]) `foaf:Person`. Property `fresnel:showProperties` takes as its value a list of URIs referencing RDF property types. In this example, properties `foaf:name` and `foaf:depiction` of resources displayed by this lens should be shown.

Basic selectors simply name the type of resources or properties that should be selected. They are easy to use but have a 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 (i.e., all resources representing persons that know more than five other persons). Such selectors, and more complex ones, can be expressed with the languages described in the next two sections.

6.2 Fresnel Selector Language

The Fresnel Selector Language (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, but tries to be as simple as possible. Still trying to avoid reinventing the wheel, FSL is strongly inspired by XPath, reusing many of its concepts and syntactic constructs while adapting them to RDF's graph-based data model. RDF models are considered as directed labeled graphs according to RDF Concepts and Abstract Syntax [**?**]. FSL is therefore fully independent from any serialization.

An FSL expression represents a path from a node or arc to another node or arc, passing by an arbitrary number of other nodes and arcs. FSL paths explicitly represent both nodes and arcs as steps on the path, as it is desirable to be able to constrain the type of arcs a path should traverse (something that

[7] Subclass and subproperty relationships must be taken into account by the selection mechanism, provided that an RDF Schema or OWL ontology is available at runtime for the associated vocabulary.

is not relevant in XPath as the only relation between the nodes of an XML tree is the parent-child relation which bears no explicit semantics).

Each step on the path, called a location step, follows the XPath location step syntax and is made of a) an optional axis declaration specifying the traversal direction in the directed graph, b) a type test taking the form of a URI reference represented as an XML qualified name (QName), or a `*` when the type is left unconstrained, c) optional predicates that specify further conditions on the nodes and arcs to be matched by this step.

The type test constrains property arcs to be labeled with the URI represented by the QName, or resource nodes to be instances of the class identified by this QName. In other words, type tests specify constraints on the types of properties and classes of resources to be traversed and selected by paths. Constraints on the URI of resources can be expressed as predicates associated with node location steps. A consequence of interpreting QName tests as type constraints is that FSL is syntactically and semantically compatible with Fresnel's basic selectors. The latter can therefore be considered a very limited subset of what can be expressed with FSL. Thus, any valid basic selector expression is a valid FSL expression.

More information about the language, including its grammar, data model and semantics can be found on the Fresnel Web site [**?**]. In the following examples, literals containing FSL expressions should all declare `fresnel:selector` as their datatype. This declaration has been omitted here for clarity.

```
# A lens for foaf:Person resources that know at least five other resources
:PersonLens a fresnel:Lens ;
    fresnel:lensDomain "foaf:Person[count(foaf:knows) >= 5]".

# Show the foaf:name property of all foaf:Person
# instances known by the current resource.
:PersonLens a fresnel:Lens ;
    fresnel:showProperties ("foaf:knows/foaf:Person/foaf:name").
```

6.3 SPARQL

The SPARQL RDF query language [**?**] offers the highest expressive power and can be used to specify lens and style domains. SPARQL queries used in this context must always return exactly one node set, meaning that only one variable is allowed in the query's SELECT clause. As with previous FSL examples, the literal's datatype declaration has been omitted.

```
# A lens for John Doe's mailboxes
:PersonLens a fresnel:Lens ;
    fresnel:lensDomain "SELECT ?mbox WHERE ( ?x foaf:name 'John Doe' )
                                           ( ?x foaf:mbox ?mbox )".
```

7 Fresnel Implementations

Note to the reviewers: Fresnel is currently being implemented by several RDF visualization tools and there are further groups that are currently evaluating it

and might descide to support Fresnel within their tools. This section reflects the current status of Fresnel implementations. We expect to update and extend this section in the camera-ready version of this paper.

Fresnel represents RDF display knowledge in an abstract and declarative manner. It provides high-level indications about how the data should be presented, but does not impose any constraint on the output format or any particular method or presentation paradigm for rendering the result. It is therefore up to the browsers and applications to interpret Fresnel display knowledge according to their own presentation paradigm. This section gives an overview about the Fresnel implementation efforts currently being conducted by several research groups.

7.1 Longwell

Longwell is a suite of RDF browsing applications written in Java and developed by the SIMILE project [**?**]. Initial releases of Longwell implemented an *ad hoc* display vocabulary, amended as application needs arose. The next major stable release, currently under development, will instead support the core selection terms from Fresnel. Graph results from searches will be filtered through Fresnel lens definitions before being passed on to the rendering engine. Also, a basic, experimental RDF browser, @@@unnamed, based off the same set of selection and browsing code, will implement all of the core Fresnel terms.

(@@@assuming a screenshot of longwell2 shows off the same data used in the IsaViz screenshot) As seen in the screenshot of Longwell **??**, the display of a `foaf:Person` using the (@@@which lens?) renders only what is specified in the lens in the order of specification; there is more data in the underlying collection involving contact information that is not shown.

Longwell generates XHTML and uses CSS to style data according to the CSS box model; this portion does not use Fresnel styling.

7.2 IsaViz

IsaViz [**?**] already implements a rendering engine that can interpret GSS stylesheets [**?**] for styling RDF models represented as node-link diagrams. The new version (currently under development) features two presentation paradigms that both support Fresnel stylesheets: the default paradigm based on node-link diagrams, and a new paradigm based on the box model described in section 6 which will take advantage of the zoomable user interface and semantic zooming capabilities associated with the graphical toolkit upon which the user interface is built [**?**].

@@@ Outline of interpretation in the node-link diagram representation

@@@ Mention alternate interpretation in the Semantic ZUI representation

7.3 Other Implementations

Another effort to implement Fresnel is the Arago RDF-browser. Arago is currently being developed at DERI and renders XHTML representations of RDF

data[**?**]. Another groups that are currently in the process of evaluating Fresnel and migh support it in their tools are the Haystack team at MIT[**?**] and the Noadster team at CWI[**?**].

8 Conclusion

Todo: Come back to abstraction of selection and styling as general problems: In this paper we have dicussed different approaches and abstraced to general prblems ... Todo: In order to ... exchange of knowledge ... we provide core-ontology for ....mention why we designed fresel that way ... browser-indpendent.....declarative...layout independent...

In this paper, we have presented Fresnel a browser-independent vocabulary for representing display knowledge for RDF data. We have demonstrated how Fresnel is interpreted by RDF visualization tools relying on different display paradigms and gave an overview about the Fresnel implementation effort currently conducted by several research groups.

Todo: Maybe mention week points (editing, lens purpose) Todo: Maybe talk about next steps.

Developing Fresnel is a community effort. We thus welcome everybody to contribute to the development of the core vocabularies, extend them with modules for special needs or support Fresnel by providing lenses and styles for their preferred vocabularies.

More information about the developments around Fresnel is found on the Fresnel website: http://www.w3.org/2004/09/fresnel-info/