

OWL-PL: A Presentation Language for Displaying Semantic Data on the Web

Technical Report (LU-CSE-09-002)

Matt Brophy and Jeff Hefflin

Lehigh University, Bethlehem PA 18015, USA
{mbb7@lehigh.edu and hefflin@cse.lehigh.edu}

Abstract. Current systems for displaying semantic data on the Web generate insufficient domain-independent views or require the adoption of complex ontological languages for display definition. We present OWL-PL, an easy to use, XSLT-inspired language for transforming RDF/OWL into XHTML for display on the Web. OWL-PL includes data selector tags inside an XHTML file, and thus encourages heavy reuse of existing technologies, such as CSS, JavaScript, and AJAX. This coupling with XHTML reduces the learning curve on traditional Web designers, while allowing for creation of the same rich interfaces that Web users have become accustomed to. The use of data selector tags in OWL-PL promotes a clean separation of raw data from document structure, similar to the separation of formatting and document structure provided by Cascading Style Sheets. OWL-PL permits the reuse of view information across ontologies using semantic inference and allows the end-user to switch between all available views on the fly. We implement an OWL-PL parser and demonstrate a functional example showing that it is easy and intuitive to create XHTML views of semantic data as well as re-create views of existing static HTML pages.

1 Introduction

Since the W3C's Recommendations of the Resource Description Framework (RDF) and the Web Ontology Language (OWL), the amount of semantic data on the web has increased significantly. As of March, 2009, the Linking Open Data project has over 4.5 billion RDF triples interlinked with approximately 180 million RDF links¹. This growth of semantically annotated data on the web enables numerous applications with capabilities not possible before. Much of the semantically annotated data on the web has been achieved by providing an RDF layer on top of an existing set of structured data. However, there is another subset of information on the web that has been much slower to evolve, the set of manually authored, relatively static web pages. For the duration of this paper, we focus on the subset of small, lightweight, and relatively static web pages in existence. In this realm of data, we see two issues preventing large scale semantic

¹ <http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/>

annotation. First, there exists a “Chicken and the Egg” problem; HTML page authors do not want to manually convert their data to RDF/OWL unless there are applications that can take advantage of it, and application developers do not want to produce semantic applications without a large set of semantically annotated data to work with [1]. Second, in the cases where HTML authors have put forth the effort to convert their data to RDF/OWL, they are forced to maintain two versions of their data: one version in HTML, for human viewing, and another in RDF/OWL, for machine interpretation. This process is tedious and error prone, allowing these two forms of data to become out of sync. We feel that addressing the second problem will be a large step towards solving the first problem.

We believe that there are three subsets of information available on the web:

- Information which only needs to be machine readable. This includes data found in robots.txt files and semantic web policy files. We believe this to be a very small subset of the data available.
- Information which either only needs to be human readable, or which the cost of adding semantic annotations is too high. This includes large sets of information, such as all of the works of Shakespeare. With further advances in technologies for automatically annotating data [1], this subset of data may decrease in size.
- Information that needs to be both human and machine readable. We argue that this set of data makes up a large subset of the information found on the web today.

The first two subsets mentioned above are currently supported by existing technologies. The first subset by existing web crawling resources, and RDF/OWL technologies for semantic web policies, and the second subset of information by technologies such as (X)HTML, CSS, JavaScript, XML, XSLT, etc. We argue that no acceptable solution has been determined for the third subset of information, that which needs to be both human and machine readable.

It is this third subset of information that forces authors to maintain two versions of their data. They must provide an RDF/OWL version for machine interpretation, and an HTML version for human consumption. One might argue that with the semantic web browsers that have been developed thus far²³⁴, any RDF/OWL data on the web could be considered human readable. While this data may be “human readable,” these browsers do not suffice as a final presentation mechanism for human viewers. Users on the web today have become accustomed to slick and attractive presentations and Graphical User Interfaces (GUI’s). Large scale adoption of the semantic web requires that it provide interfaces at this level or better. Providing authors a mechanism to easily display their RDF/OWL data in a visually appealing way will eliminate the need to

² <http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/>

³ <http://brownsauce.sourceforge.net/>

⁴ <http://www.w3.org/2005/ajar/tab>

maintain two versions of data, and promote conversion of existing HTML data into RDF/OWL.

A correctly designed solution, in addition to facilitating the display of RDF data on the web, should also provide authors with an additional layer of separation. In developing web-based GUI applications today, there is a strong emphasis on separating content from formatting. This argument has been relevant since the creation of the WWW and has fueled the development and enhancement of (X)HTML and CSS. Figure 1 describes the separation in traditional Web development, in which (X)HTML should be used to determine what information to display, while CSS should be used to decide how to display it. (X)HTML encodes the raw data with a document level structure, while CSS formats and styles the document. With a mechanism to transform RDF/OWL data into XHTML, we remove the raw data from the XHTML file, thus providing a third layer of abstraction and making changes to an individual piece easier.

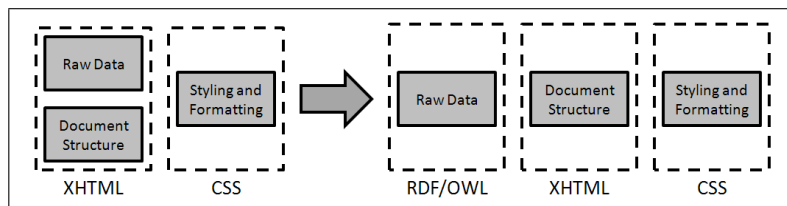


Fig. 1. Separation of concerns in traditional Web development versus our proposed separation in semantic web development.

Previous work has been done to display RDF/OWL data on the Web [2–8], but we feel that no existing technologies address all of the requirements of the different people involved. Current solutions for displaying semantic data on the web resort to creative methods for browsing and searching data, however, final user views of instances remain mostly as domain-independent table-based layouts. Furthermore, existing methods for designing enhanced views are complex, involve a steep learning curve, and do not leverage existing Web technologies appropriately. A satisfactory solution to this problem must satisfy the needs of the ontology author, the instance author, the designer of the human view, and the human viewer. In most cases, we feel that individuals will play more than one of these roles, but we define our requirements at the role level. We propose the following requirements for an acceptable solution to the problem:

Ontology/Instance Author

- The author of the ontology or the instance shouldn't need to be concerned with a human view of their data. Their main focus should be correct definition of the data and its structure. Authors should be minimally affected by the solution.

- When interested, however, authors should be able to define default presentation views for their classes and/or instances.

Designer of the Human View

- Views should be reusable across ontologies and instances, although not necessarily across different domains.
- A view should normally be domain-dependent, but ontology-independent. Human viewers do not want to look at all types of data through the same view. A different view should be presented to a human user when looking at data about a person rather than data about a music CD. However, views should be ontology-independent, so that music CD's in different ontologies can use the same view. This ontology-independent view of data can be achieved through the use of semantic inference and mapping ontologies.
- It must be easy and intuitive to display property values of an instance in various ways.
- Views must be able to traverse the RDF graph, and "jump" between different nodes.
- The view should allow incorporation of existing technologies where possible, including XHTML, CSS, JavaScript, and XSLT, to ease the learning curve for view designers.

Human Viewer

- The interface presented to the end user should possess the same look and feel as current web interfaces. This includes displaying data in a format relevant and useful to the viewer's needs, and making use of multimedia (images, videos, etc.) as well as existing GUI technologies, such as Asynchronous JavaScript and XML (AJAX).
- The end user should be able to override the default view for a given instance. This could be either by specifying a new view, or by switching to other views indicated by the ontology, the instance, or the view itself.

We present OWL-PL, an easy to use, XSLT-inspired language for transforming RDF/OWL data into XHTML on-the-fly for human viewing through a normal web browser. With its use of data selector tags inside an XHTML file, the OWL-PL language introduces the third layer of separation mentioned above, providing a clean method to separate raw data, document structure, and display information. This coupling with XHTML promotes heavy reuse of existing technologies such as XHTML, CSS, and JavaScript and eases the learning curve on the designer of the view. Additionally, due to the heavy reuse of existing technologies, OWL-PL allows for designers to create the same rich interfaces that end users have become accustomed to. OWL-PL permits the reuse of view information across ontologies using semantic inference and allows the end-user to switch between all available views on the fly. OWL-PL also allows the modularization of small view templates by including a construct for view to embed additional external views. We also demonstrate an implementation of OWL-PL

using the VisuOWL parser, a Java Web Application capable of parsing OWL-PL files with respect to OWL instances. We show that it is easy and intuitive to create XHTML views of semantic data as well as re-create views of existing static HTML pages. The rest of this paper is organized as follows: In Section 2, we present related work for displaying RDF/OWL data on the web. In Section 3.3, we discuss the OWL-PL Language. In Section 4, we present the VisuOWL parser and provide real world examples of displaying existing RDF/OWL data as well as converting existing HTML data into RDF/OWL format, while retaining the current HTML presentation. Finally, in Section 5, we conclude and discuss planned future work on the OWL-PL language and VisuOWL parser.

2 Related Work

As stated earlier, much work has already been done to address the issue of human viewing of semantically annotated data. The majority of these works fall into three broad categories: RDF browsing, ontological display definition, and XSL-influenced display definition. Additionally, there is a fourth area of work involving associating existing XHTML data with RDF. In the following sections, we discuss each of these approaches, and briefly review related work in these areas.

2.1 RDF Browsing

The category of RDF browsing refers to domain-independent tools that provide a way of navigating through RDF graphs, and a domain-independent method for viewing an instance once located in the graph. Several works have been published in recent years promoting systems and techniques for browsing RDF data on the web.

Noadster aims to provide “simple but domain-independent hypermedia generation from RDF stores” [2]. Using a search based interface, users are able to query an RDF store to generate a list of matching RDF-annotated URI’s. These URI’s are presented in a hierarchical view in the global interface. Subgroups of a common parent are sorted based on the number of matching resources, in essence listing groups with the most relevant content earlier in the list. Side-by-side with this global interface of all matching URI’s is the local interface. The user is able to click on any result in the global interface to choose this as the focal point, and generate a local view of this resource. The local view is a table based structure of all associated statements of a focal point, where an associated statement is any statement including the focal point as either the subject, predicate or object. One useful feature to point out is that this local interface does not show only outlinks from a given node, but inlinks as well. Noadster enhances the local interface by displaying `rdfs:label` properties in bold as the title of the page, and using MIME types of non-literals (URI’s) to determine how to display the external media referenced by the URI.

Due to the semantic information present in RDF/OWL documents, faceted browsing seems a logical choice for navigating such data. BrowseRDF is prototype of a faceted interface for RDF data [3]. By treating RDF subjects as information elements, RDF predicates as facets, and RDF objects as restriction values, BrowseRDF presents an interface for exploration of unknown data. This exploration is possible because the system is able to suggest all possible restriction values at each step in the process, and is able to restrict the user from choosing “dead-ends,” or facets that do not have any available restriction values. The system’s basic selection is the selection of a direct restriction value. Enhancing the usefulness of the system is the presence of existential selection (does a resource have this property), join selection (selections on properties of related nodes), intersection of various selections, and inverse selections. Furthermore, BrowseRDF uses a clever facet ranking metric in order to sort facets in order of relevance to the user, based on predicate balance, object cardinality, and predicate frequency. As a user chooses facet values, they are presented with a basic list view of the resources that match the given facet values, and the property values of these resources. BrowseRDF presents a very useful manner in which to navigate or find a given resource, but does not focus on useful ways to display that resource once located.

Many RDF-based visualizations have a tendency to treat RDF as a graph, and allow for systems to navigate the graph, however, in their position paper [9], Karger and Schraefel, ask (1) Are graphs the right *default* presentation for the Semantic Web, and (2) If not, how might we decide on a default presentation such that the benefits of knowledge building and sharing are available to users. Karger and Schraefel claim the pathetic fallacy of RDF is that “researchers are (perhaps subconsciously) allowing the computer’s internal representation of data to influence the way their tools present information to users.” The authors present the following issues with presenting RDF data as a BFG:

- Graphs are flat, and treat all nodes equally, regardless of a current focal point. Additionally, there is no concept of a primary node versus a secondary node.
- Graphs do not scale well in large datasets.
- Graphs represent items and nodes and links, but a user’s informational need may very well be in separate areas of the graph. Viewing data that is not in close proximity does not work well in a graph.
- They fail to address any domain-dependent layouts which assist human users in better understanding information.

One very insightful question posed by the authors is, “What, if anything, is special about what the Semantic Web enables such that existing UI paradigms do not suffice?” The authors further argue that, to the extent that we have currently found the “right” interfaces in the WWW, we should continue to use those same interfaces to display semantic web data. This point has been an important point driving the development of OWL-PL.

2.2 Ontological Display Definition

This category refers to domain-dependent tools that allow creation of ontologies that define appropriate views of instances adhering to a given ontology. Bizer, Lee, and Pietriga [4] promote the distinction between two major questions in any environment in which information is displayed to an end user. First, what should we display (content selection), and second, how should we display it (content formatting and styling). This argument relates directly back to the separation of concerns in the traditional web, which we discussed in the introduction. Advocates of ontological display definition refer to this argument to support their decision to encode style information in ontologies, separate from ontological data. We argue that this ontological encoding of document level structure and styling information is complex, and that we should leverage existing technologies such as CSS and XHTML as to separate document level structure from styling information, while moving raw data out of XHTML and into RDF/OWL files. In the remainder of this section, we discuss some of the works related to ontological display definition.

Bizer, Lee, and Pietriga present Fresnel [4], an ontological presentation language. Fresnel provides ontology and instance authors the ability to create ontologies describing the proper presentation format for the RDF-encoded data. To address the two concerns above, Fresnel incorporates the concept of *lenses* and *formats*. *Lenses* allow the user to specify which data to display from an RDF document and in what order, while *formats* allow the user to specify how to display the selected data (as a link, as an image, etc.). The authors also present another very important point in that these concepts of “what to display” and “how to display it” should be declared in a manner that allows reuse between browsers, applications, and ontologies. While Fresnel does provide the advantage of being reusable across different display paradigms (HTML, PDF, SVG, etc.), we argue that the formatting options available to the author are too generic as a result, and individual implementation is left up to the application incorporating Fresnel. This may remove too much control from the author’s hands when deciding how to display their data. We also argue that the process of creating an ontology to specify what and how to display data is an unnecessary abstraction. With the broad acceptance and usage of technologies like (X)HTML, XSLT, and CSS, we should aim to reuse these technologies where possible, to ease the learning curve on authors.

In their paper [5], Quan and Karger also present the same argument for separating content from presentation, and present Xenon, an RDF stylesheet ontology. The authors advise reuse of many of the key ideas of XSL Transformations (XSLT), however they chose to implement their language in RDF. The Xenon ontology provides many of the functions available in XSLT, however they are specified and linked to existing RDF data through ontological properties, which results in a very complex document describing the presentation of a resource. Xenon addresses the separation of concerns question with the concept of *lenses* and *views*. *Lenses* will display properties from a set of properties that make sense being shown together, such as a name lens, or a summary lens. A

lens is able to be specified across different resources (such as a name lens for a book and for a person) as well as indicate the expected size of their generated output (one line, full screen, etc.). *Views*, on the other hand, are abstractions that signify that a set of resources should be displayed in some manner, but delegate the responsibility of displaying the individual resources. For example, a view may define a method to display a list of Person resources, but may indicate that each Person resource should be displayed in one line. Using this specification, the Xenon parser will look for and choose a lens that claims to display a Person resource on a single line. This framework allows for display details to be determined at run-time, which is a nice feature. Quan and Karger provide the following benefits to using RDF to encode style sheet information:

- The use of RDF ensures that the style sheet information is as universal as schema or ontology information.
- No need to force users to adopt a specialized syntax, as existing syntaxes such as RDF and N3 can be used.
- RDF has a built in mechanism for “merging” RDF documents, and thus the semantics of combining two style sheets is clearly defined.

We argue that these three benefits can be provided through an XML based XSLT-like syntax because (1) XML is accepted as a universal format, (2) XSLT is widely accepted, and thus conforming to an XML-based language similar to XSLT will reduce the users learning curve, and (3) XSLT templates provide functionality to reuse view information, although not automatically. We argue that automatic merging of two RDF style sheets is not in the user’s best interest. Styling information in one stylesheet may be irrelevant or less valuable when automatically merged with or embedded in a separate stylesheet. We feel that the use of an XSLT template-like approach, in which the author would specify merge information (what documents, which is the parent, where does the merge occur), will maintain the intended display of the merged stylesheet.

Quan and Karger present another work that is relevant to our topic of ontological display. In their paper, Quan and Karger describe Haystack [6], a *Semantic Web* browser, which the authors differentiate from a *Semantic* web browser. One notable difference about Haystack is that it runs as an application on the user’s local machine, rather than through a web interface. Haystack uses the same abstractions of *lenses* and *views* as the Xenon stylesheet ontology discussed above⁵. Provided by default in Haystack are two lenses applicable to any instance. The All Properties lens shows every property for the given instance, and the Standard Properties lens shows a set of properties from the Dublin Core ontology, such as title, and author/creator. For more advanced views and lenses, Haystack uses an internally developed View Ontology Web Language (VOWL) ontology, an internal Haystack ontology, as well as existing ontologies such as DAML+OIL and Dublin Core. By including these properties in RDF data, Haystack is provided with the information it requires to generate views

⁵ Actually, Quan and Karger developed Haystack prior to Xenon, and thus Xenon reuses the lenses and views defined for the Haystack system.

of the instance. Using an RDF store abstraction, Haystack permits RDF statements to be added and removed from heterogeneous data sources, and provides an event mechanism such that when new properties are added to the store, the interface is notified and can update accordingly. These are admirable concepts that should add to the usefulness of the final user view. Additionally, Haystack uses Adenine [7], a data definition language and imperative programming language to encode advanced user views of semantic resources. Although Haystack presents the most advanced and visually appealing interfaces we have seen thus far, we argue that the methods for generating these views, from including properties from multiple different ontologies to writing code in the Adenine language, increases the learning curve for view designers and will reduce the acceptance of this application as a viable solution to the problem. Additionally, the implementation of Haystack as a local application running on the user's computer may reduce acceptance among general users who can be reluctant to download and install external applications on their local machines.

2.3 XSL-influenced Display Definition

The third category consists of tools that adapt the XSL family of languages to support RDF graphs, rather than simple XML documents. These tools allow an RDF-based version of XPATH to query RDF data, and use a transformation language such as XSLT to transform data into a form for presentation on the web. First, we feel that it is important to address XPATH and XSLT in their current form, and discuss their shortcomings with respect to RDF/OWL documents.

At a first glance, XSLT seems to be an appropriate solution to the problem of appropriately displaying semantically annotated data on the web. XSLT, combined with XPATH, provides a way to extract specific information from an XML document and transform it into (X)HTML for presentation on the web. RDF/OWL files are valid XML documents, and thus could be queried via XPATH, so the shortcomings of XSLT are not immediately apparent. However, there are several shortcomings of XSLT and XPATH with respect to OWL/RDF documents:

- RDF/OWL documents do not follow a rigid XML structure. Two documents may be vastly different from an XML/XPATH perspective, yet still convey the same semantic information. Querying these documents with XPATH breaks down due to these structural concerns. Steer provides a good example of this in his presentation of TreeHugger⁶.
- Information for a given RDF/OWL instance may not be located in a single file. XPATH would break down when `<owl:imports>` statements were present.
- XSLT is incapable of processing inferred information, as this is not explicitly contained in the initial RDF/OWL document.

⁶ <http://rdfweb.org/people/damian/treehugger/introduction.html>

It is clear that the XSL family of languages is not the solution to our problem; however, we argue that these languages provide the correct framework and should have a major influence on the eventual solution.

In order to combat the shortcomings of XPATH above, proposals surfaced for a modified version of XPATH, often referred to as RDFPath⁷⁸⁹. Many of these proposals arose in the past, but haven't yet been adopted, and many other proposals now point to dead links on the web. Implementations we have seen do agree on a striped implementation, where each subsequent step in the RDFPath query switches between a node and an edge (e.g. an `rdf:class` and `rdf:property`). The Fresnel project mentioned above uses their own XPATH-influenced language for querying RDF documents, the Fresnel Selector Language for RDF (FSL)¹⁰. Arago [8] implements FSL in PHP5, providing the ability to generate views defined using the FSL.

We argue that the XSLT approach to defining displays for RDF data is appropriate, and this is the category OWL-PL falls into. As we will discuss in Section 3.3, OWL-PL is an XSLT-influenced display definition language, with a simple method for accessing data concerning a given instance, without the complexity of an RDFPath language.

2.4 Associating RDF and XHTML

The final category of tools we discuss tackle the problem from the opposite angle used in the XSLT-like approaches discussed in Section 2.3. Rather than using an XSLT-based approach to generate (X)HTML from RDF, these works propose methods for authors to associate their XHTML with RDF which allows these files to be transformed into RDF data for machine consumption. RDFa¹¹ uses the extensibility of XHTML and adds a new set of valid attributes that can be used to specify that certain data in an XHTML document refers to an RDF resource or property value. Using these attributes, the RDF data can be extracted and constructed into an RDF view of the data. However, we feel that associations to support the full spectrum of the RDF language would be overly complex for XHTML designers.

The Gleaning Resource Descriptions from Dialects of Languages (GRDDL)¹² takes a similar approach; however, GRDDL does not embed RDF annotations directly into the XHTML file. Rather, an XHTML file specifies an external GRDDL transformation file, usually encoded in XSLT. When this transformation file is applied to the static XHTML file, the resulting RDF document is produced. One advantage of using an external transformation file is the ability to publish reusable transformations for common micro formats. We feel that this XSLT

⁷ <http://rdfweb.org/people/damian/treehugger/introduction.html>

⁸ <http://infomesh.net/2003/rdfpath>

⁹ <http://logicerror.com/RDFPathProposal>

¹⁰ <http://www.w3.org/2005/04/fresnel-info/fsl>

¹¹ <http://www.w3.org/TR/xhtml1-rdfa-primer/>

¹² <http://www.w3.org/TR/grddl/>

approach is tedious, as the XSLT files to generate an appropriate output are likely to be very long and complex. Furthermore, these XSLT files are heavily dependent on the document structure of the XHTML file, and will need to be updated when the XHTML file changes.

We argue that both of these works address the problem from the wrong direction. As discussed in Section 1, in the current web, we encode both raw data and document structure in XHTML. In order to enhance the user experience, we organize the raw data as to achieve the best document structure (for human viewing), not to achieve the best data or ontological structure (for machine consumption). It is flawed to believe that we can extract the appropriate ontological structure from a format focused on document structure. Instead, we must focus first on correct semantic encoding of the data, and then on appropriate document structure for the data.

3 OWL-PL Language

OWL-PL is a presentation language for transforming RDF/OWL data into XHTML for display in existing web browsers. OWL-PL operates on the subset of RDF data contained in OWL DL. OWL-PL consists of two main concepts: a formatting ontology and the OWL-PL language. Here we present both of these concepts, along with a theoretical discussion of requirements.

3.1 Formatting Ontology

In order to specify how a given instance of a class should be displayed, we need an ontology that handles these properties. For this reason, we present the formatting ontology in Figure 2 (namespacing and entities left out for readability). In this ontology, we define properties allowing instances and classes to identify OWL-PL format files that are appropriate for displaying the instance, or instances of the given class. The *hasFormat* property allows any instance to specify the URI's of OWL-PL format files which can be used to display that instance. The *hasDefaultFormat* property is a sub-property of *hasFormat*, and is intended to be a single valued property identifying the default format to be used for the instance. The *hasAltFormat* property is used to identify any additional format files which could be used to display the given instance. Similarly, the *hasDefaultClassFormat* annotation property allows a class to specify the URI of an OWL-PL format file which should be used as the default display for any instances of that class, while the *hasAltClassFormat* annotation property is used to specify additional format files which can be used to display instances of the class. When displaying an instance, the order of precedence for choosing a format file is: *hasDefaultFormat*, *hasDefaultClassFormat*, *hasAltFormat*, *hasFormat*, *hasAltClassFormat*. This allows instances to override any class specifications using *hasDefaultFormat*, while also allowing instances to supplement any *hasDefaultClassFormat* properties using *hasAltFormat*. We use defeasible inheritance semantics for the *hasDefaultClassFormat* and *hasAltClassFormat* properties, so

that values associated with more specific classes override those of more general classes. Additionally, in our ontology, we have given the class `owl:Thing` a *hasDefaultClassFormat* property value pointing to a simple table based display of property values. This ensures that any instance that does not specify a format file will eventually inherit one from `owl:Thing` if no other ancestor classes specify a format file.

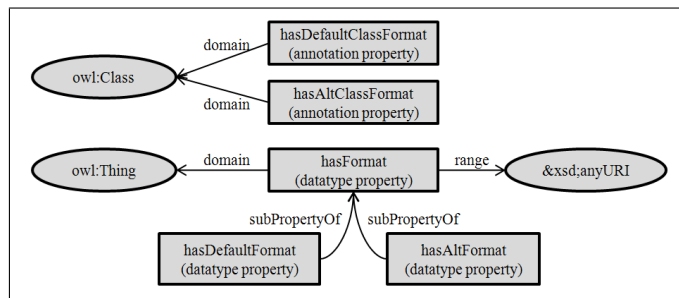


Fig. 2. OWL-PL Formatting Ontology.

A few points should be stated concerning the chosen design of this ontology:

- Format files default to a simple table-based layout inherited from `owl:Thing`
- Format files are then overridden at each child sub-class, using any additional *hasDefaultClassFormat* properties.
- Format files are further overridden at the instance level, by any *hasFormat* properties on the instance.
- The viewing user can override at the last step by specifying a format file when viewing the instance.
- All format files located in this hierarchy are presented to the end user in the final display, and the end user is given the option to switch between different views.

3.2 Abstract Model

We define the following theoretical application. We propose a function *render()* accepting four inputs: the URI of an RDF/OWL document *d*, an optional instance *i* in *d*, an optional view *v*, and an optional mapping ontology *m*:

$$render(d, [i], [v], [m])$$

Given the inputs, a knowledge base KB consisting of a T-Box and an A-Box can be constructed by taking the imports closure of *d* and importing any mappings defined in *m*. We can parse the view *v* with respect to KB and produce an XHTML view of the instance *i*, or all instances in *d* if *i* was not specified.

The matching of any conditions specified in v is based on entailments of the KB (as determined by a DL reasoner). The view to construct, if not specified, will be determined from the data available within KB. It will determine the most specific view by looking first at the instance data for a specified view, and then by traversing up through parent classes, and defaulting to a table-based view if no views are specified by the instance or its ontologies.

Given this framework, authors can create ontologies and data and publish them to the web. If they choose, they may also develop views of this data, and specify them in the ontology or the data. Further, separate designers can create views of this data, and make them available. Finally, a human viewer chooses a document (or instance) to view and is presented with the default presentation for the instance(s). If the human viewer chooses, they can specify a different view to use to display the instance. If this view happens to be for a different ontology, then the human user has the ability to specify a mapping ontology, which maps the properties of the instance to the properties used in the view. Algorithm 1 outlines a pseudocode algorithm for the *render()* function. The following additional functions are used within the render function:

- *readIndividualsFromDocument(d)* - This function returns a set of all individuals specified in the document d .
- *gatherImports(d)* - This function returns a set of all axioms in ontologies imported in document d .
- *materializeSubsumptions(KB)* - This function computes all subsumptions in KB using a DL reasoner, and makes them explicit inside the knowledge base KB.
- *getFormat(i, KB)* - This function returns the appropriate format file following the order of precedence specified in Section 3.2. When searching for the value of *hasDefaultClassFormat* and *hasAltClassFormat* properties, we implement defeasible inheritance reasoning by performing a breadth first search from the most specific to the most general source classes of the instance. Note, since these properties are Annotation Properties, they are ignored by the DL reasoner.

3.3 OWL-PL Formatting Language

The OWL-PL formatting language is very closely related to the existing XSLT language for transforming XML documents. As stated in Section 2.3, we believe that the existing XSL family of languages provides the correct framework for transforming RDF/OWL data for presentation on the web, provided it can be appropriately adapted to support the additional semantic information provided by the RDF/OWL languages. Despite the shortcomings noted in Section 2.3, some of the current features of XSLT are directly useful when applied to RDF/OWL data, and we have mimicked them very closely.

An OWL-PL format file is XHTML with embedded OWL-PL data selector tags, indicating what data to display from a given instance. In the abstract

Algorithm 1 *render*(*d, i, v, m*)

```
if i ≠ null then
  individuals = { i }
else
  individuals = readIndividualsFromDocument(d)
end if
KB = d ∪ gatherImports(d) ∪ FormattingOntology
if m ≠ null then
  KB = KB ∪ m
end if
materializeSubsumptions(KB)
for i ∈ individuals do
  if v = null then
    v = getFormat(i, KB)
  end if
  constructView(v, i, KB)
end for
```

model above, an OWL-PL format file serves as the view of a given instance. It is important to note that while parsing an OWL-PL format file, we are always focused on a single instance in the knowledge base. Although this instance may change during the processing of the format file, as you will see shortly, there is always a single instance of focus. Very similar to XSLT, certain OWL-PL nodes carry special meaning when parsed in the XHTML file, and replacing these nodes and their children with transformed nodes results in the final XHTML file to display to the user. Figure 3 shows a very short example of a format file that prints the last and first name of an individual.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
[ <!ENTITY P "http://www.cse.lehigh.edu/someontology.owl#" > ] >

<owlpl:format xmlns:owlpl="http://www.cse.lehigh.edu/owlpl/">
  <p style="font-weight: bold;">
    <owlpl:value-of select="&P;lastName"/>, <owlpl:value-of select="&P;firstName"/>
  </p>
</owlpl:format>
```

Fig. 3. Sample OWL-PL Format File

The following is a brief overview of the XSLT inspired nodes in the OWL-PL formatting language, preceded by an introduction to some of the terms used in

the overview. Additional detail concerning the nodes is available in the OWL-PL online documentation¹³.

Terms

- **propertyUri**: This refers to the full URI of a given property. The property can be either a datatype or object property
- **stringOrProp**: This refers to a string in which special braced sections will be treated as property URI's and replaced with their corresponding values. The string `prefix{propertyURI}suffix` will result in `{propertyURI}` being replaced with its corresponding value, and `prefix` and `suffix` remaining unaltered.
- **objectPropertyURI**: The full URI of an object property
- **formatURI**: The full URI of an OWL-PL XHTML format file

XSLT-inspired Nodes

- `<owlpl:value-of select="[propertyUri]"/>`

For the focused instance, display the first value for the property given by the "select" attribute. The `rdfs:label` property is displayed for object properties if available, rather than the URI of the referenced instance.

- `<owlpl:for-each select="[propertyUri]" [focus-on="y"]> ... </owlpl:for-each>`

Process the children nodes for each property value of the property specified in the "select" attribute. If the specified property is an object property and the "focus-on" attribute is set, then parse the children nodes relative to the new property instance. Otherwise, one may use an `<owlpl:value-of>` child node to print a multi-valued datatype property.

- `<owlpl:if lhs="[stringOrProp]" op="[op]" rhs="[stringOrProp]"> ... </owlpl:if>`

Only evaluate the XHTML child nodes if the expression indicated by the "lhs," "op," and "rhs" attributes evaluates to true. The "op" attribute accepts `<`, `<=`, `==`, `!=`, `>`, and `>=` for numerical comparisons, and `lt`, `le`, `eq`, `ne`, `gt`, and `ge` for string comparisons. For example, the following will evaluate to true when processed according to my RDF instance, assuming that `&P;` is the entity for my ontology, and I have a property `&P;lu.user.id` with a value of `mbb7`:

```
<owlpl:if lhs="{&P;lu.user.id}@lehigh.edu" op="eq" rhs="mbb7@lehigh.edu">
```

¹³ <http://ikkin.cse.lehigh.edu:8080/owlpl/doc>

```

- <owlpl:choose>
  <owlpl:when lhs="[stringOrProp]" op="[op]" rhs="[stringOrProp]">...</owlpl:when>
  ...
  <owlpl:otherwise> ... </owlpl:otherwise>
</owlpl:choose>

```

This node allows for a switch-case statement, and the `<owlpl:when>` node follows the same processing rules as the `<owlpl:if>` node above. The XHTML child nodes of the `<owlpl:otherwise>` are only parsed if none of the `<owlpl:when>` nodes evaluate to true.

```

- <owlpl:useFormat format="[stringOrProp]" />

```

This node maps loosely to the `<xsl:apply-templates>` node, and provides a mechanism for reusing existing format files. These can be specified using the *hasFormat* property discussed in section 3.1 or through a specific URI of a format file.

It is apparent that by replicating some of the current XSLT functionality, and replacing XPATH queries directly with property references, we can already handle some basic transformation and display of OWL data. Now, we will briefly discuss some additional nodes available in the OWL-PL formatting language that are not based on existing XSLT functionality.

Additional OWL-PL Nodes

```

- <owlpl:for-all> ... <owlpl:prop/> ... <owlpl:value/> ... </owlpl:for-all>
  <owlpl:for-others> ... <owlpl:prop/> ... <owlpl:value/> ... </owlpl:for-others>

```

These nodes provide a convenient way to dump all of the properties of the focused instance. The child tree of these nodes will be parsed for each property/value pair, with the `<owlpl:prop/>` node being replaced with a property URI (localname) and the `<owlpl:value/>` node being replaced with the value of the property. When the `<owlpl:for-others>` node is used, it will only display those properties of the focused instance not yet displayed on the page.

```

- <owlpl:focus-on select="[objectPropertyURI]"> ... </owlpl:focus-on>

```

This node provides a convenient mechanism to jump to a related node. The children of this node will be processed relative to the instance pointed to by the *select* attribute. As in the `<owlpl:value-of>` node, this node chooses the first property value, and should be nested within an `<owlpl:for-each>` node for multi-valued object properties.

— `<owlpl:link [uri="[stringOrProp]" [format="[stringOrProp]"]> ... </owlpl:link>`

The child tree of this node will be wrapped in a normal XHTML `<a>` tag providing a link to view the instance given by the "uri" attribute, optionally displayed with the format file given by the "format" attribute.

— `<owlpl:otherFormat uri="[formatUri]"/>`

This node can be used in a format file to indicate other format files that are appropriate to display this type of instance. These nodes are only interpreted from the root format file, and not from any additional format files included using an `<owlpl:useFormat>` node. The reasoning for this is that the external formats used to display a portion of the page may not specify format files appropriate for displaying the entire current instance, and thus we do not present them to the user as viable alternatives.

— `<a owlpl:href="[stringOrProp]">`

Any non OWL-PL-specific node will be checked for attributes prefixed with "owlpl:" and if found, these attribute values will be handled as brace-syntax OWL-PL strings, and replaced accordingly. The "owlpl:" prefix is then removed after processing. An example usage of this is as follows, assuming "&P;" is the XML entity of my ontology, and I have an appropriate email address specified using the datatype property "&P;email:"

```
<a owlpl:href="mailto:&P;email">Email Me!</a>
```

— `<owlpl:annotate with="[objectPropertyURI]" on="[stringOrProp]"/>`

This node can only be used as a child of an `<owlpl:value-of>` node for a datatype property, but allows us to annotate the literal property value with links to instances referenced within. It is best explained through an example. Let's assume that we are presenting a news article that references John Doe, an instance in our ontology. Assuming the news article has an object property `&P;references` that points to John Doe's instance, then:

```
<owlpl:value-of select="&P;articleText">
<owlpl:annotate with="&P;references" on="{&P;firstName} {&P;lastName}"/>
</owlpl:value-of>
```

will wrap any occurrences of the text string 'John Doe' (generated from the firstName and lastName properties of John Doe's instance) in the datatype property `&P;articleText` in normal XHTML links (`<a>`) that will take the user to a page to view John Doe's instance in the OWL-PL browser.

4 Application and Examples

In this section, we present the VisuOWL parser, a Java Web application capable of parsing OWL-PL format files and producing XHTML for display in a normal web browser. We demonstrate real-world examples for generating a view of an existing semantic web resource using an OWL-PL format file as well as transforming an existing HTML view of static data into the same OWL-PL view of semantic data.

4.1 VisuOWL Parser

The VisuOWL parser is implemented as a Java Web Application, using Servlets and JSP technologies. On the back end, VisuOWL uses the HAWK Toolkit¹⁴, in order to access ontology and instance data and to support inference using a DIG interface to a DL reasoner. Because OWL-PL format files are required to be valid XHTML files, our VisuOWL parser uses a DOM algorithm to process the XHTML tree. Following the determination of the appropriate format file to apply to an instance, as explained in Section 3.1, the parser operates recursively through the DOM tree, replacing special OWL-PL nodes and attributes as they are encountered. In order to increase efficiency of the processing, ontology files, data files, and format files are cached for the duration of the user's session, and thus only need to be read and parsed into a DOM structure once per session.

4.2 Examples

There are two main areas in which we see the OWL-PL and VisuOWL browser being useful. First, the VisuOWL browser should be able to display existing RDF data, with little work required by ontology authors. Second, the OWL-PL language should make it easy for authors of static HTML pages to reproduce those interfaces using semantically annotated data. We experimented in both of these areas and present our results here.

In order to test our first anticipated usage of the VisuOWL browser, the display of existing RDF/OWL data, we leveraged the Linking Open Data project. We chose the RDF Book Mashup as our existing data set and chose the RDF document describing Tim Berners-Lee's Weaving the Web book¹⁵. Feeding this URI into the VisuOWL parser, without specifying any format file or mapping ontology, we are given a basic table layout of all of the properties for this instance. We are quite satisfied that without any additional work by the ontology author or instance author, we as the end user are able to view existing RDF data in a manner very similar to much of the existing works discussed in section 2.1. Furthermore, with a 21 line format file (8 lines to define entities, and only 13 to define the presentation), we were able to create and specify a format file to use to display the RDF document in a much more user friendly manner (Figure 4).

¹⁴ <http://swat.cse.lehigh.edu/projects/index.html#hawk>

¹⁵ <http://www4.wiwiss.fu-berlin.de/bookmashup/doc/books/006251587X>

We note that neither of these approaches required any change to the ontology or instance data, just a creation of an XHTML format file in the second example.

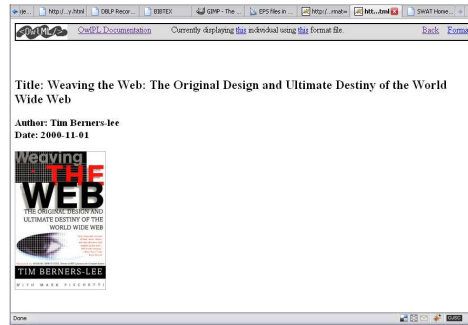


Fig. 4. OWL-PL enhanced views of existing RDF data

In order to test our second main usage of the OWL-PL language and the VisuOWL parser, we aimed to recreate most of an existing, static HTML webpage using semantically annotated data. We chose the existing web page for the SWAT lab at Lehigh University¹⁶. This static HTML webpage is maintained by members of the SWAT lab, and contains information pertaining to the people, projects, publications, and events related to the SWAT lab. As shown in Figure 5, using the OWL-PL language and the VisuOWL parser, we were able to very closely replicate the existing HTML version of the SWAT website, yet do this from a completely semantic version of the data.

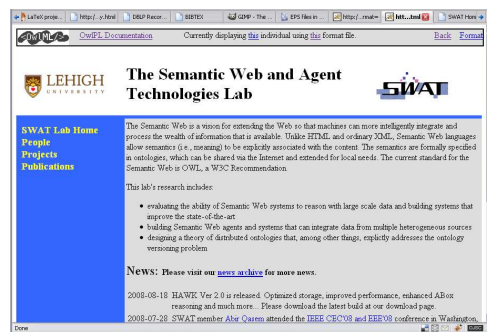


Fig. 5. Re-created OWL-PL version of the SWAT webpage

¹⁶ <http://swat.cse.lehigh.edu/>

We will now go through a few code examples of the format files used to generate the view from the SWAT Lab ontology. In order to replicate this information using OWL-PL, we developed an ontology to describe the data. In the following sections, we will provide relevant sections of the research ontology along with extracts of code from the format files used to recreate the SWAT website. Note that in all examples the entity `&R;` has been defined to point to the research ontology, and excessive HTML nodes, attributes and CSS styles have been removed for readability.

Header Using the `<owlpl:useFormat>` node, we are able to take reusable sections of a page, such as header data, and move them into separate format files for reuse. Figure 6 shows the properties and values that will be used in this example. In the header of the SWAT Lab web page, we first focus on the `&R;subOrganizationOf` property to switch the focus from the Lab to its parent organization (lines 4-8). While focused on the organization, we use the `owlpl:` attribute prefix to generate a hyperlink to the web page of the organization (line 5) and then again to include the organizations logo (line 6). We note that these attribute prefixes use the brace syntax discussed in section 3.3, in which braced sections of the string are treated as property URI's and replaced with their corresponding values, and thus the string `prefix{propertyURI}suffix` will result in `{propertyURI}` being replaced with its corresponding value, and `prefix` and `suffix` remaining unaltered. After reverting focus back to the Lab, we use the `<owlpl:value-of>` node to display the title of the lab (line 11), and then the `owlpl:` attribute prefix again to display the logo of the Lab (line 14).

```

1. <owlpl:format xmlns:owlpl="http://www.cse.lehigh.edu/owlpl/">
2.   <tr>
3.     <td>
4.       <owlpl:focus-on select="&R;subOrganizationOf">
5.         <a owlpl:href="{&R;hasWebPage}">
6.           
7.         </a>
8.       </owlpl:focus-on>
9.     </td>
10.    <td><h1>
11.      <owlpl:value-of select="&R;hasTitle" />
12.    </h1></td>
13.  </td>
14.    
15.  </td>
16. </tr>
17. </owlpl:format>

```

Menu As shown in Figure 7, the *SwatLab* instance specifies datatype properties identifying additional format files that can be used to display different aspects of the SWAT lab. The menu for the website resides in a separate file, for reusability, and is included from all other views. Here, for space concerns, we only display the links in the menu file. The key idea is that different pages about similar data in existing HTML pages are achieved by applying different views to the same

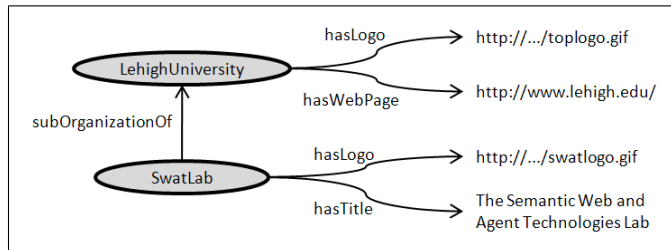


Fig. 6. Ontological properties used in the SWAT header format file

semantic instance using OWL-PL. Here, we provide views of the *SwatLab* where each view is focused on a specific aspect of the lab (it's people, projects, and publications). This is accomplished using the `<owlpl:link>` node to provide links that switch the format file used to display the instance (lines 1-4). Note, if no document or instance is specified (as in this example), the current document and instance are used by the `render()` function.

1. `<owlpl:link format="{&0;hasDefaultFormat}">SWAT Lab Home</owlpl:link>
`
2. `<owlpl:link format="{&R;hasPeopleFormat}">People</owlpl:link>
`
3. `<owlpl:link format="{&R;hasProjectFormat}">Projects</owlpl:link>
`
4. `<owlpl:link format="{&R;hasPublicationFormat}">Publications</owlpl:link>`

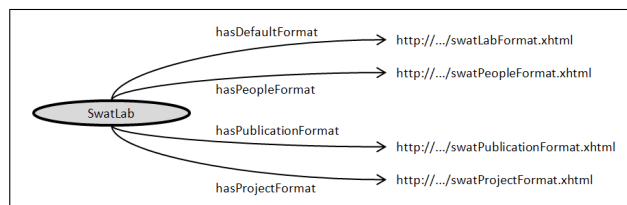


Fig. 7. Datatype properties indicating additional format files for the SWAT Lab.

Figure 8 shows the subset of the Research ontology that will be used in the remaining examples.

Lab Members This snippet is taken from the page displaying all of the lab members. On this page, we want to loop through all of the people in the lab, and display them in sections based on what type of person they are (Director, Alumni, Graduate Student, etc.). Because, at the ontological level we have a single property *hasMember* that links a *Person* to a *Lab*, we cannot have specific loops for each type of person. Instead, we must loop through all *hasMember* property values (line 3), and check their type using an `<owlpl:if>` node (line

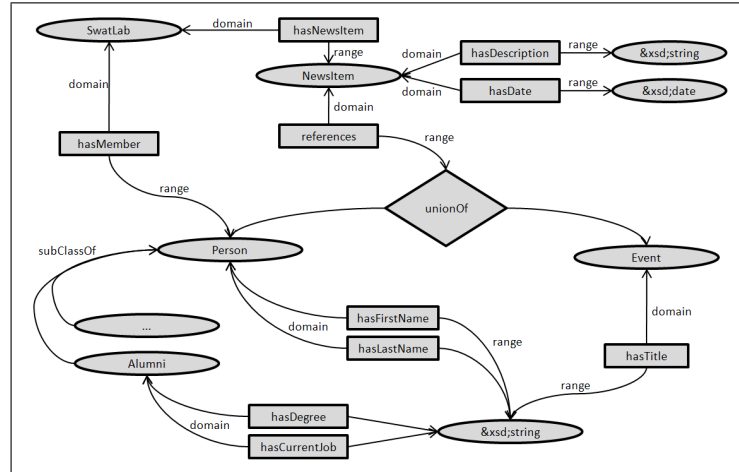


Fig. 8. Subset of the Research ontology. The diamond construct is shorthand for a node with a datatype="collection".

4). Note that the `<owlpl:for-each>` node includes the "focus-on" attribute, so the children nodes are parsed according to the each *Alumni* instance as we loop through. Thus, the `<owlpl:value-of>` nodes (lines 7, 8, 9, 10) are printing data from the *Alumni*, not the *Lab*. We note that on line 10, we have included the "default" attribute with a blank string. This is used to make sure that nothing is printed if no property value is found. Without this attribute, the VisuOWL parser will display 'No value specified.' Finally, we use the `<owlpl:link>` node (line 6) so that each *Alumni* is a clickable link that will take us to view that instance. Note that here we have chosen to not specify a specific format file for the link; rather, we allow the format file to be determined from the instance data, or inherited from its parent classes. Figure 9 shows the corresponding section of the resulting page.

```

1. <p class="bold">Alumni:</p>
2. <ul>
3.   <owlpl:for-each select="&R;hasMember" focus-on="y">
4.     <owlpl:if lhs="{&RDF:type}" op="eq" rhs="&R;Alumni">
5.       <li>
6.         <a owlpl:href="mailto:{&R;hasEmailAddress}">
7.           <owlpl:value-of select="&R;hasFirstName"/>
8.           <owlpl:value-of select="&R;hasLastName"/></a>
9.           (<owlpl:value-of select="&R;hasDegree"/>),&nbsp;
10.          <owlpl:value-of select="&R;hasCurrentJob" default=""/>
11.        </li>
12.      </owlpl:if>
13.    </owlpl:for-each>
14. </ul>

```

We compare this to what would have been required if we chose to allow OWL-PL to use a SPARQL query to extract this information from the RDF



Fig. 9. Alumni section of the resulting SWAT lab people page

document. The following SPARQL query would provide the information required to generate the above view of alumni members:

```
PREFIX research: <http://.../swatOnt.owl#>
        rdf: < http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?fName, ?lName, ?degree, ?job
FROM <http://.../swatData.owl>

WHERE
{
  <http://.../swatData.owl#swatLab> research:hasMember ?person.
  ?person rdf:type research:Alumni.
  ?person research:hasFirstName ?fName.
  ?person research:hasLastName ?lName.
  ?person research:hasDegree ?degree.
  ?person research:hasCurrentJob ?job.
}
```

We feel that this is overly complex, and that it is easier and more intuitive using the `<owlpl:for-each>` and `<owlpl:value-of>` nodes. Additionally, in order to include a SPARQL query such as this inside an XHTML OWL-PL format file, one would need to encode the appropriate XML entities (`<`, `>`) with their equivalent entity representations (`<`, `>`), which would further increase the complexity of the file. For this reason, we have decided that using SPARQL queries is not an appropriate method for extracting ontological data for display. In section 5, we discuss future possibilities of using an RDFPath-like language, such as the Fresnel Selector Language, to extract information in a more concise manner.

News Items Our final example is the most complex, so we will discuss it in further detail. Many times while browsing the web, we find long blocks of textual content, such as news stories or blog entries. It is commonplace for these entries to contain hyperlinks to specific key entities referenced in the block of text, thus allowing the user to follow the links to learn more information concerning the entity. Many times, users will open these links in separate windows, and view the two items concurrently for additional detail or understanding. Due to separation of concerns, it is not appropriate to include these hyperlinks inside of RDF literals, such as news stories or blog entries, however, we feel that it

is not unreasonable to expect that RDF textual content will reference external entities. In these cases, we see the need to be able to *annotate* textual RDF data with links to RDF resources referenced by the text.

From the ontology overview above, one can see that the object property *hasNewsItem* relates the SWAT lab to *NewsItems* concerning the lab. Furthermore, the object property *references* indicates what *Events* or *People* the *NewsItem* relates to. Knowing this information, we can use the `<owlpl:annotate>` node to annotate the *NewsItem* with links to the *People* or *Events* that it references. First, we loop through all *NewsItems* about the lab (line 1), and switch focus to these instances. Then we print the date and descriptions of each of these instances (lines 3, 5). We then choose to annotate the description of the news item (line 6). This line and the preceding line indicate that we want to annotate the *hasDescription* property of the *NewsItem* with any instances pointed to by the *references* property. By default, we will annotate substrings matched on the *hasTitle* property of the referenced instance. Thus, if we find any *People* or *Events* whose *hasTitle* property value occurs in the *NewsItems hasDescription* value, we will wrap that text with an HTML link to view the *Person* or *Event* referenced. Lines 7-9 provide an override, stating that if the referenced instance is of type *Person*, then we should instead match the *hasFirstName* and *hasLastName* property values of the person, separated by a space. The resulting image (Figure 10) shows the final displayed *NewsItem*, in which two people were referenced, as well as one event. It should be noted, that in the current implementation, these matches are required to be exact matches. In the future, we plan on including a mechanism to specify a matching threshold to allow for partial matching.

```

1. <owlpl:for-each select="&R;hasNewsItem" focus-on="y">
2.   <tr>
3.     <td width="80"><owlpl:value-of select="&R;hasDate"/></td>
4.     <td>
5.       <owlpl:value-of select="&R;hasDescription">
6.         <owlpl:annotate with="&R;references" on="{&R;hasTitle}">
7.           <owlpl:if lhs="{&RDF;type}" op="eq" rhs="&R;Person">
8.             <owlpl:match on="{&R;hasFirstName} {&R;hasLastName}"/>
9.           </owlpl:if>
10.         </owlpl:annotate>
11.       </owlpl:value-of>
12.     </td>
13.   </tr>
14. </owlpl:for-each>

```

2008-08-04 SWAT director [Jeff Heflin](#) and member [Abir Qasem](#) attended the [IEEE-ICSC 2008](#) conference in Santa Clara, CA. Abir presented "Efficient Selection and Integration of Data Sources for Answering Semantic Web Queries" in the conference.

Fig. 10. Resulting display of annotated *NewsItem*

We point out that all of the examples above use a generic research ontology, and thus if another research lab at a different university used a different ontology, they are able to reuse these format files by mapping to this research ontology. Ontologies, data, and format files used in these examples can be found on the Examples page in the OWL-PL online documentation¹⁷.

5 Evaluation, Conclusions and Future Work

We have shown that it is possible to use an XSLT-influenced presentation language to generate human readable XHTML pages of RDF/OWL data on the fly. For a simple evaluation metric of the work required to generate an XHTML view of semantic data, we have compared the number of lines in the existing SWAT HTML web pages versus the number of lines required in our OWL-PL format files to generate a comparable presentation. This is not quite a straightforward comparison due to the combination of raw data and document structure in the static HTML pages. The four static HTML pages that we recreated totaled 665 lines of HTML (including raw data). We extracted all of the raw data from the existing HTML pages and created an ontology (229 lines) and data file (498 lines) that describe the same raw data. We then created the corresponding OWL-PL XHTML format files to generate these same four pages, which totaled to 305 lines of XHTML with embedded OWL-PL tags. We note the fact that the header and menu format files are reusable on each of the SWAT pages (people, publications, etc.), and thus these lines need only be counted once, while in the existing HTML version, these lines are duplicated in each HTML page. If we assume that the ontology and data would need to be maintained in parallel with the existing static HTML pages, then we do not count the lines required for the semantic markup of data, and we have cut the total number of lines from 665 lines of HTML to 305 lines of XHTML in the OWL-PL format files, a 54% reduction in the number of lines. If we assume that there was no corresponding semantic version of the data, then we require a 55% increase in total lines, mostly due to the verbose nature of the RDF/OWL XML syntax. Although this seems to be a prohibitively large increase in lines, we argue that creation of this semantic version of the data allows for the use of this data in other semantic applications.

We believe that OWL-PL provides a much simpler method for generating rich interfaces than existing methods, which produce insufficient domain-independent views, or require the adoption of overly complex ontological languages. Furthermore, OWL-PL encourages the heavy reuse of existing Web technologies, such as XHTML, CSS, and JavaScript. This simplicity of creation and the emphasis on XSLT similarity and reuse of existing technologies will ease the learning curve for new ontology and view authors, and promote the semantic annotation of existing data presented in static HTML pages. The concept of data selector tags in the OWL-PL language provides a clean separation of raw data from document structure, thus easing the individual maintenance of data

¹⁷ <http://ikkin.cse.lehigh.edu:8080/owlpl/doc/examples.jsp>

and structure. The flexibility of the OWL-PL language, in its ability to reuse view information across ontologies using semantic inference, embedding of external view templates, and allowing the user to switch between views on the fly, helps address the individual needs of the ontology and instance authors, the view designers, as well as the end user. We believe that providing authors an easy method to generate views of semantic data that are comparable with current web interfaces with respect to ease of use and visual status will work hand in hand in promoting the semantic annotation of existing static HTML pages. This, in turn, will increase the amount of semantically annotated data on the web, and drive the development of powerful semantically aware applications.

This paper describes the very early stages of the OWL-PL and VisuOWL projects, and we believe there is still much work to be done. One main concern is the efficiency of the system. The bottleneck in the system currently is split between the loading of specified RDF/OWL documents, loading any ontology files imported by these documents, and parsing the OWL-PL XHTML format files. We are currently using a DOM parser for the format files, and are considering switching to a SAX parser to increase efficiency. Furthermore, we would like to improve the caching mechanism. Currently, ontological data and format files are cached at the session level; however there may be reason to use a longer term and globally available data store for this information, thus allowing caching across users and sessions. We also foresee this type of caching as possibly suggesting new format files for a given instance, based on data it has seen previously. We are also interested in modularizing the system to allow for different methods of data retrieval, such as the Fresnel Selector Language (FSL). This would move beyond the simple selection techniques currently employed, and would provide additional functionality such as sorting and counting of properties. We would like to devote more time towards improving our error handling techniques. Due to the potential inaccuracy or absence of data in the semantic web, we feel that a system needs to be robust, and needs to fail gracefully in the absence of certain data. We would like to implement a method to make use of partial matches when annotating textual RDF data. This feature would expand the areas in which annotation can be useful. Finally, with the ability to reuse format files across ontologies and instances, we would like to further investigate the complexities presented by JavaScript and CSS. Without proper precautions in the VisuOWL system and by the view designer, formatting (CSS) and user interaction code (JavaScript) may conflict with or be overridden by external format files if not namespaced appropriately. Additionally, external media, such as JavaScript files, CSS stylesheets, and images using relative paths will need to be appropriately de-referenced when imported to a separate format file.

Acknowledgements Thanks to Bryan Auslander and David Heefner for their assistance during the prototype implementation of the OWL-PL language and VisuOWL parser.

References

1. Cimiano, P., Ladwig, G., Staab, S.: Gimme' the context: context-driven automatic semantic annotation with C-PANKOW. In: 14th International World Wide Web Conference. (2005) 332–341
2. Rutledge, L., van Ossenbruggen, J., Hardman, L.: Making RDF presentable: integrated global and local semantic web browsing. In: 14th International World Wide Web Conference. (2005) 199–206
3. Oren, E., Delbru, R., Decker, S.: Extending faceted navigation for RDF data. In: Fifth International Semantic Web Conference. (2006) 559–572
4. Pietriga, E., Bizer, C., Karger, D.R., Lee, R.: Fresnel: A browser-independent presentation vocabulary for RDF. In: Fifth International Semantic Web Conference. (2006) 158–171
5. Quan, D., Karger, D.R.: Xenon: An RDF stylesheet ontology. In: 14th International World Wide Web Conference. (2005)
6. Quan, D., Karger, D.R.: How to make a semantic web browser. In: WWW, New York, NY, USA, ACM (2004) 255–265
7. Quan, D., Huynh, D., Karger, D.R.: Haystack: A platform for authoring end user semantic web applications. In: Second International Semantic Web Conference. (2003) 738–753
8. Gassert, H., Harth, A.: From graph to GUI: Displaying RDF data from the web with Arago. In: Second European Semantic Web Conference/Scripting for the Semantic Web Workshop. (2005)
9. Karger, D., Schraefel, M.: The pathetic fallacy of RDF, Third International Semantic Web User Interaction Workshop (2006) http://swui.semanticweb.org/swui06/papers/Karger/Pathetic_Fallacy.html. Accessed 06/09/2009.