EROS: A User Interface for the Semantic Web

Richard Vdovjak       Peter Barna       Geert-Jan Houben
Computer Science Department
Eindhoven University of Technology
PO Box 513, Eindhoven, The Netherlands
+31 40 2475199
{richardv,pbarna,houben}@win.tue.nl

ABSTRACT
To bring the advantages of the Semantic Web to the end-user, one clearly needs an interface which would allow the user to explore, and query the content she is interested in. RDF(S) is an acknowledged backbone of the Semantic Web architecture. We argue that due to its peculiarities a new interface metaphor is needed to convey RDF(S) based ontologies to the end-user in a comprehensible form. We present a framework for browsing and querying RDF(S) based ontologies in the context of a photo library application, which serves as our running example.

Keywords: EROS, User Interface, RDF(S), Semantic Web, Ontologies, Photo Library

INTRODUCTION
It seems that the current focus of the Semantic Web activities lies mainly in defining machine readable and machine understandable standards like XML [1] and RDF(S) [2, 4] respectively. These foundations are certainly needed to accommodate the ultimate Semantic Web vision where (software) agents discover, negotiate, and collect relevant information crawling from one Web source to another. However, we are currently far from this vision and the main consumers of the Web information are still humans. The technologies introduced by the Semantic Web activity can turn websites into knowledge bases where end-users can expect simply a better service when it comes to responding to their information needs.

Despite very good results provided by current keyword-based search engines, a query consisting of a set of keywords is sometimes not enough. For instance, when relationships among the keywords play an essential role, one clearly needs a more expressive mechanism than just plain keywords.

To illustrate this, consider the following query: “Find a picture of the US part of the Niagara Falls taken from the Falls Avenue in the town Niagara Falls (the Canadian bank) with a lens of 400 mm or longer”. The first part of the query denotes the subject (waterfalls) being photographed. The second identifies the position of the photographer. Note the ambiguity of the “Niagara Falls” collocation, first denoting the waterfalls, and then being a part of the position description as a town name. Moreover, there is the third part of the query, imposing an additional lens constraint which basically says that we are only interested in those images that provide enough details and a narrow perspective achievable only by using a lens of that focal length.

It is evident that queries of this kind are not likely to be satisfactorily answered by keyword based search engines. By translating this query into a set of keywords and trying a keyword based search engine we either obtained an empty set of results (e.g. Google Image Search) or a countless number of irrelevant pictures featuring big photo lenses (e.g. AltaVista Image Search retrieved over 1.4 million images and the top scored ones were indeed mostly showing long lenses and other photo equipment).

Examples like this show a clear need for something more powerful than keywords. The use of ontologies, taxonomies of classes within a certain domain linked by properties indicating different relations among those classes, would enable to enhance queries and improve both the precision and the recall. The technology for defining such ontologies is already available. RDF(S) defines a standard language for defining simple ontologies. There are authoring tools that help the designer during the process of ontology creation. There are also different query languages and their implementations which allow for querying such ontologies. Yet there are very few applications that combine these technologies into a user-friendly environment, so most of the end-users are still left with keyword-based searches.

We acknowledge that the work that has been done so far on actual creation of semantic annotations of Web resources is not sufficient. However, this work has to start incrementally in small communities related to specific domains, which will agree (or already agreed) on the terms they are using and find the right motivation to write them formally down to ontologies, and allow others to benefit from them as well. The more people will be using these ontologies the more popular they will become and will evolve, interconnect with other “neighbouring” ontologies and eventually create something that we could call the emerging Semantic Web. So, apart from creating ontologies, one of the essential prerequisites to make the Semantic Web happen is also to convince the end-users to take advantage of already existing ontologies for accomplishing their tasks. To do so, they must be offered a whole framework of tools wrapped nicely by a user interface that is both expressive and comprehensible enough. Here it is where we see the major gap between what is currently available and what the end-user desires. We try to address these issues by our framework described in the paper.

The rest of the paper is structured as follows. We start with a description of RDF and RDFS followed by a section which
introduces our running example and compares existing RDF user interfaces. In the next section we introduce the EROS$^1$ interface and in the subsequent section we show how it can be used as a front-end for querying our photo ontology example. The last section concludes the paper and sketches the outline for the future work.

BACKGROUND: RDF AND RDFS

The Resource Description Framework (RDF) [4] is a general-purpose language issued as a W3C standard. While relying on XML for serialization, RDF focuses on semantics and lays a foundation for processing metadata across the Web.

An RDF model consists of resources, named properties, and property values. Property values can be either literals (simple strings or other primitive data types defined by XML) or again resources, which in turn may have (possibly) other properties attached to them. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. If we represent RDF resources and literals as nodes labeled by URIs$^2$ and literal values respectively (yet providing a syntactical distinction between the two) and RDF properties as edges labeled with property names, an RDF model turns into a directed labeled graph (DLG) with the following properties.

- It allows for multiple edges between two nodes. However, there are no outgoing edges from literal nodes.
- Nodes with the same label are mapped into one node, with the exception of so-called blank-nodes which do not have a label and each occurrence of such a node is considered to be unique.
- Different edges between two nodes cannot have both the same label, and orientation. In other words, a triplet consisting of two nodes connected by an edge appears in the graph just once.
- The graph can contain cycles.
- The graph can be composed of several, not necessarily connected (sub)graphs.

RDF Schema (RDFS) [2], an extension to RDF, which is itself expressed in RDF terms, provides a support for creating vocabularies at the type (schema) level. RDFS defines a modeling language by assigning a special semantics to several resources and properties, amongst which the most important are:

- The resource rdfs:Class is the root of the type system that any newly defined class/type must be an instance of (rdfs:Class is also an instance of itself).
- The resource rdfs:Property represents the class of all properties.
- The property rdf:type indicates the instance membership.
- The transitive property rdfs:subClassOf / rdfs:subPropertyOf provides an inheritance-like mechanism for building hierarchies of classes / properties. A class / property is allowed to have several of these properties, which means it can have several super classes / super properties (similar to multiple inheritance).

- The properties rdfs:domain and rdfs:range are applied to properties (instances of the class rdfs:Property) and define their domain and range respectively. The use of multiple domain / range properties is allowed and is interpreted as saying that the resulting domain / range is an intersection of all the specified properties / classes.

Note that properties are associated with classes only by specifying their rdfs:domain and rdfs:range. That facilitates the defining of properties separately from classes making them first-class citizens and facilitating one of the architectural principles of the WWW that anyone can say anything about existing resources.

THE CHALLENGE IN VISUALIZING RDFS ONTOLOGIES

We motivated in the introduction why it is important to have a good user interface (UI) for RDFS-based ontologies$^3$. Now, what is so challenging in designing such a UI? The problem in our opinion lies in the fact that it is difficult to show the whole expressive power of RDFS and at the same time to keep the user interface still comprehensible, easy to use, browse and navigate. In the following subsections we introduce the RDFS based ontology of our running example and discuss the UI metaphors currently used by existing applications.

The Running Example: Photo Stock Library.

To design a photo stock library that would allow users to express their semantic queries over the stocked images, one needs an ontology capturing the relevant terms within the given domain. Similarly to [9], our ontology example consists of several sub-ontologies, whose terms are distinguished by different namespaces.

- The photo ontology consists of terms coming from the photo domain. It describes things like different kinds of Light, various photo Techniques, different camera Settings etc. The cornerstone of this ontology is the class Photo which is linked by its properties with the aforementioned classes. There is also a property called depictstheme that connects the Photo class from the photo ontology with Entity class from the general ontology.

- The general ontology consists of all terms one can possible take a picture of, the most general term being the Entity class.

- The ternary ontology serves as a means to describe a story, where there are two or more actors that perform (either send or receive) an action. For instance a photo depicting a man biting a dog is certainly a different story than that depicting a dog biting a man [5].

---

1 Explorer for RDF(S)-based OntologieS
2 Universal Resource Identifiers
3 For practical reasons we focus mainly on visualizing the schema level.
The Graph Metaphor
Since an RDF based ontology is in fact a graph with labels on edges and nodes, the most straightforward approach is to mimic this structure by a UI, which itself is a full-fledged directed labeled graph. This paradigm certainly captures most of the expressive power of RDFS. However, for ontologies with more classes it becomes almost unreadable for the end-user, overwhelming him/her by its complexity.

![Figure 1: Graph based UI of KAON [6].](image)

The most sophisticated, graph-based user interface we have tried so far is the one implemented within the KAON project [6]. Except of the standard graph features it offers several add-ons like gradual exploration (unfolding) of rdfs:subClassOf and rdfs:subPropertyOf hierarchies and the “self-adjusting” algorithm that arranges the appearance of the entire graph by changing the position of nodes and edges each time the user goes deeper in the hierarchy or modifies the graph otherwise. While this feature works fine when applied to a graph consisting of a few nodes, it becomes very slow and not very interactive if one wants to explore a graph with more nodes. Moreover, once the graph is unfolded it is difficult to grasp the hierarchical structure which is “hidden” behind the special edges and not reflected by the position of the class nodes. This, together with the fact that the graph tends to grow uncontrollably in both dimensions, are the major drawbacks of most graph-based UIs. Figure 1 shows the KAON graph interface displaying our ontology example, illustrating some of the aforementioned issues.

The Tree Metaphor
In a tree-like UI, unlike in the graph metaphor, nodes are organized strictly hierarchically, usually by the property rdfs:subClassOf. or when the property-centric view is more appropriate the tree is created following the rdfs:subPropertyOf property. This has several positive but also negative implications.

On the positive side is the fact that the tree UI metaphor is very common and its behavior is well understood since the end-users are familiar with browsing trees of different kinds (e.g. deep directory structures, bookmarks, mail folders etc). Unfolding or browsing a tree can be controlled much better both in vertical and horizontal dimension compared to unfolding or browsing a graph. For instance, in the case of a tree UI, each level down the hierarchy moves all ancestors of the current node with a constant distance to the left or to the right. However, unfolding a node in a graph can cause a need to rearrange the whole graph by a variable (hard to predict) expansion in both dimensions.

![Figure 2: Tree based UI of Protégé [7].](image)

The drawbacks of the tree approach originate mainly from the fact that an RDFS based ontology is still a graph and there is a discrepancy between a rigid tree structure and a general graph. In other words it is hard to capture a graph in a tree. For instance, it is difficult in a tree-based UI to depict things like multiple inheritance and also properties that (arbitrarily) associate classes to other classes or to literals. Figure 2 shows a tree-based UI of Protégé-2000 [7], an ontology editor, which we used for creating our ontology example. The Protégé UI consists of two major parts.

The left part depicts the class hierarchy by a browsable tree below of which there is a list of super-classes of the currently selected node.

The right part mainly consists of a list of properties (Template Slots) together with the (textual) description of their ranges, i.e. class names or primitive types, to which these properties point. While this approach is quite suitable for the design of ontologies, it becomes rather cumbersome for the process of browsing, exploring or familiarizing with an existing ontology, mainly because it is impossible to see at the same time both the domain class together with its context (i.e. the position in the class hierarchy) and the range class together with its context.

THE EROS METAPHOR
Combining the advantages of both metaphors, one would desire the simplicity of the tree based approach and at least a part of

---

4 KArlsruhe ONtology and Semantic Web infrastructure.
5 Besides the graph interface the KAON framework offers also a tree-like preview.
6 This is the way Protégé handles multiple inheritance.
the expressiveness of the graph based approach. This is exactly what we tried to achieve by the EROS interface depicted in Figure 3.

Class Centric Approach

The main idea behind this interface is to consider properties as partial mappings that map (some) elements (classes) from the class hierarchy into either other (possibly identical) elements within the same hierarchy. Note that the set of all elements from the hierarchy serves two purposes: firstly as a (potential) domain of all properties and later as their (potential) range. This double purpose inspired us to actually have two (almost) identical hierarchy trees in our interface, the left tree being the domain ("from") tree, and the right tree being the range ("to") tree. Properties themselves are depicted as arrows connecting the classes from the domain tree with the classes from the range tree. Note that this approach makes it possible to display for a certain property at the same time both the context of the domain class and the context of the range class. Cases of multiple inheritance are handled the same way as in the Protégé UI, i.e. for a currently selected class in the tree the interface offers a list of its super classes.

Figure 3: EROS UI: Class Centric Approach.

To avoid overcrowding the interface we adopted the following display strategy. If a selected node has more than three properties, their names are displayed on demand as a "hint text", evoked by the user’s mouse pointer. In case there are several different properties connecting two classes, we represent all of them with one bold arrow offering a context menu consisting of the list of the "hidden" properties, so that the user can specify which property is used to traverse from the domain class to the range class.7

Property Centric Approach and User Adaptation in EROS

The approach described in the previous section admittedly favors classes over properties as the rdfs:subClassOf property is the key for building the tree hierarchy. However, one may prefer to view the ontology with the "property optics" and desire to explore the tree hierarchy based on the property rdfs:subPropertyOf. The philosophy of the EROS interface can easily accommodate this demand only by imposing that the domain tree is built based on the rdfs:subPropertyOf relationship instead of the rdfs:subClassOf relationship. The rest stays intact. The user now has a view where in the left tree there is a hierarchy of properties connected by their rdfs:domain and rdfs:range properties to the tree on the right which still contains the hierarchy of classes.

![Figure 4: EROS UI: Property Centric Approach.](image)

In fact, EROS goes even further and allows the user to adapt the interface to his needs by choosing an arbitrary transitive8 property as a key for building the left tree and another one for building the right tree. In this case one can, for instance, display the resources of his ontology in the left tree based on the wordnet:hyponym relation and in the right tree on the rdfs:subClassOf property and study the (possibly subtle) differences in these tree hierarchies and also their mutual relationships in terms of properties which connect resources from one tree to another. Especially in issues like integrating or aligning ontologies, this ontology mapping is a complicated process and EROS partially offers support for this.

QUERYING RDFS BASED ONTOLOGIES

To facilitate the use of an existing ontology and its instances (annotations), two prerequisites are essential. Firstly, the users have to become familiar with the ontology; here a good and flexible UI such as EROS is certainly of a great help. Secondly, there must be tools that enable users to express queries over the ontology, execute them, and return resulting instances (resources). When it comes to querying RDFS, RQL [3] is one of the most advanced query languages to date. It covers both queries over RDF Schema and RDF Instances. RQL queries, similarly to those from SQL, consist of SELECT FROM WHERE clauses. The SELECT clause specifies

---

7 This becomes important when the user wants to define a path expression, which is the main building block of RQL queries.

8 Transitive from the user’s point of view, not only from the built-in point of view of RDFS.
resources (variables) which are of interest. The FROM part is the core of the query and specifies one or more path expressions (i.e. subgraphs of the entire graph) where the variables are bound. The WHERE clause contains filtering conditions that are being applied on the variables bound in the FROM statement.

Despite its relative intuitiveness, RQL is still too low-level for an average end-user who wants to formulate queries over RDF-based ontologies. The query creation part of the EROS interface can however guide the user while composing an RQL query. This process consists of filling in all the three clauses in a point-and-click manner.

As the most difficult clause to build is the FROM part, the EROS interface starts by assisting the user in the formulation of a single or chained path expression of the form:

\[
\{VAR_1 : DC_1 \}_p \{VAR_2 : RC \lor L \}_p \{VAR_3 : RC \lor L \}_... \\
\{VAR_n : DC_n \}_p \{VAR_n : RC \lor L \}_... 
\]

Where \(VAR_i : X\) denotes a variable of type \(X\) and \(p\) represents a property, which has as a domain the class \(DC\), and as its range either the class \(RC\) or \(L\) (a literal). EROS allows the user to generate such expressions by selecting a node in the graph (a variable of a certain type), then selecting a property \(p\) of the chosen node that navigates the user to the destination node (again a variable). If the destination node is not of a literal type, the user can choose another property of that node traversing the graph further (building a chained path expression) or s/he can create a new path expression concatenating it with the previous ones.

Filling in the SELECT clause consists of choosing variables that are of interest from the list of variables introduced in the FROM clause.

Similarly to the SELECT clause, the WHERE clause starts by selecting variables bound in the FROM clause and then building a Boolean expression by utilizing an offered list of appropriate operators.

One should note that the queries that can be built in this way are among the more simple ones (i.e. we do not support nested queries or implicit schema queries), but they do represent a large class of practical queries for ontology exploration. As an example, even these simple RQL queries can express the kind of user queries we mentioned in the introduction. Concretely, the following is an RQL equivalent of the user query "Give me all pictures of Niagara Falls taken from the Falls Avenue in Niagara Falls, Canada with a lens of 40 mm or more".

**SELECT PHOTO**

**FROM** (PHOTO : GeoPhoto) depictsTheme (THEME : Universe) (PHOTO) takenFromPlace (PLACE : AddressablePlace) . country (COUNTRY) . (PLACE) town (TOWN) , (PLACE) street (STREET) . (PHOTO) takenWithSettings . focalLength (LENS)

**WHERE** THEME like "Niagara Falls" and COUNTRY like "Canada" and TOWN like "Niagara" and STREET like "Falls Ave" and LENS >= 400

Note that this query is actually a copy of the generated query from the EROS interface depicted in Figure 5.

**Putting it All Together**

To put the EROS interface at work we created a framework that implements our photo stock library example. The framework (depicted in Figure 6) consists of two parts; the client side and the server side.

The client is represented by the EROS Java-Web application downloaded to the user's computer. Here the user explores the photo ontology and builds a query, which is then sent to the server part. The results are coming as a browsable HTML presentation of relevant images.

The server part contains the Sesame RQL engine [8], running as servlet on the Apache webserver, which hosts also the actual images and another servlet (Sesame Wrapper) that wraps the required functionality of the Sesame engine, serving as a mediator between Sesame and the EROS UI. This mediator also contains a simple RDF2HTML XSLT stylesheet that transforms the results of the query from the Sesame-RDF format into an HTML presentation. Our photo ontology together with its instances is stored in a MySQL database, generated by Sesame.

**CONCLUSIONS AND FUTURE WORK**

As the nature of applications changes under the influence of the Semantic Web (SW) initiative, the need to capture the semantics of the application data increases. In the typical application representing SW ideas, the semantic annotation of Web resources is crucial. As an illustrative example we showed how
keyword-based querying does not suffice, and how RDF(S) can help to improve the query process. Essential in the use of RDF(S) are the ontologies. They help to organize the resources in terms of their semantics, and thus offer a nice specification of the semantics of the entire application. However, for many applications this specification in terms of ontologies needs to be used by humans. They use such an ontology to find or search for terms and to mentally reason about these terms. Before they are constructing actual queries on the RDF metadata they need to familiarize with the ontology. For this purpose an effective visual representation of ontologies is vital. We have considered existing proposals for tools offering visual representations of ontologies, based on the two metaphors of graph-based and tree-based. We have considered the advantages and disadvantages of both metaphors and shown how the ideal combination for a large part of the user requirements is a new metaphor that offers the simplicity of the tree based approach and at least a part of the expressiveness of the graph based approach. In the EROS interface the user is able to view the ontology both from the viewpoint of classes or that of properties in terms of their domain and range classes. We have shown how this can help in general, but specifically it supports the creation and manipulation of queries on RDF metadata. As an example we have shown how the EROS interface assists in the generation of RQL queries. The EROS metaphor proves to be helpful in the process of conceptually “designing” (a large subset of) RQL queries, while the EROS interface actually facilitates the implementation of this query definition process.

The current version of EROS is being improved in order to allow more refined RQL queries. Although the current version supports already a large subset of queries, that is enough for the purpose of practical metadata querying, we want to increase the expressive power of EROS while maintaining the strength of its metaphor. Another issue for current and future work is that of the implementation of the gradual loading of voluminous ontologies.

Currently, the EROS interface loads the whole model (the schema) at once, but this is not ideal and moreover not needed in most practical cases. The solution that we are devising is based on a navigation-driven exploration/processing of voluminous RDFS ontologies when the user is “diving” into a node by sending the appropriate RQL (class) queries, we can have a gradual retrieval of the node’s subnodes.

REFERENCES


