Bringing the “Wiki-Way” to the Semantic Web with Rhizome

Adam Souzis[1]

[1] Liminal Systems, 4104 24th Street Ste. 422,
San Francisco, CA, USA
asouzis@users.sourceforge.net
http://www.liminalzone.org

Abstract. The Wiki and the Semantic Web can be compared as two different approaches to capturing knowledge, where the former trades away precise, explicit, and internally consistent semantics for speed and simplicity. Any attempt to bridge these two approaches has to either somehow reconcile these trades-off or make compromises one way or the other. This paper describes how Rhizome, an open source application framework for developing “Semantic Wiki” applications, attempts to bridge these approaches. Rhizome includes a text formatting language called ZML whose syntax is similar to text formatting languages found in most Wikis but with enhancement to make it easy for users to express explicit and arbitrary semantics. Rhizome relies on “shredding”, a flexible framework for specifying rules for characterizing semi-structured content with RDF and providing an ontology that can precisely describe the relationship between the source content and the resulting statements.

1 Background

The Wiki and the Semantic Web can be compared as two different approaches to capturing knowledge, where the former trades away precise, explicit, and internally consistent semantics for speed and simplicity. Any attempt to bridge these two approaches has to either somehow reconcile these trades-off or make compromises one way or the other; for example, by adding complexity and constraints that undermines Wiki design principles or by limiting the scope where Semantic Web data can be applied (e.g., limiting it to meta-data associated with traditional wiki pages).
The Wiki has proven to be a remarkably successful tool capturing knowledge in a collaborative, open fashion. The inventor of the Wiki, Ward Cunningham, has identified several Wiki design principles, which he refers to as the “Wiki-way”[1]. A review of his descriptions of some of these principles is suggestive of how they can be challenging for applications that utilize and create Semantic Web data:
“**Mundane** – a small number of (irregular) text conventions will provide access to the most useful page markup”[2] But this approach doesn't easily lend itself to making precise and controlled statements; indeed Semantic Web scenarios generally assumes a specialized user interface for a particular application domain.
“**Unified** – Page names will be drawn from a flat space.”[2] This principle seems in accord with the use of universally unique URIs as the basis of names for the Semantic

web; however, the scope of this namespace is so huge it is pragmatically difficult to treat as a flat space.
“**Tolerant** – Interpretable (even if undesirable) behavior is preferred to errors.”[2] But ontologies and ontologies languages generally require some degree of internal consistency to function properly.
“**Open** – any reader can edit [a page] as they see fit.”[2] However, when the content being created is Semantic Web data which can be readily consumed by -- and alter the behavior of – applications, security concerns must be addressed.
This paper attempts to conform to the ABCDE format for Semantic Conference Proceedings[3]; the next section, “Contribution” describes how Rhizome[4], an open source application framework that makes it easy to develop “Semantic Wiki” applications, contributes to the challenges outlined above; this is followed by the Discussion section which describes Rhizome's architecture in more depth.

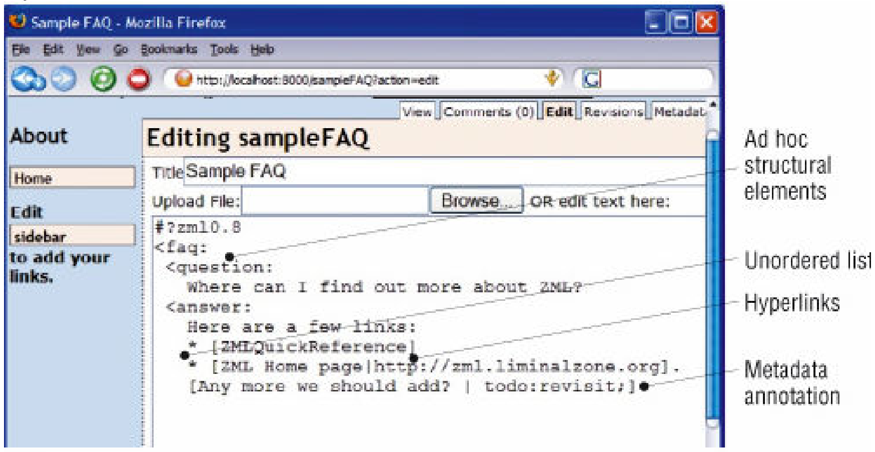
2. Contribution

Rhizome is an open source application framework that makes it easy to develop “Semantic Wiki” applications: applications that can create and utilize RDF data and Semantic Web ontologies while letting users interact with and modify that data in a Wiki-like fashion. In this section we describe how Rhizome attempts to fulfill the Wiki design principles discussed above.

2.1 Mundane

What sort of “(irregular) text conventions” should be used for authoring RDF triples? The simplest approach would be a text format limited to providing a way to explicitly describe RDF triples. And arguably, existing plain text RDF formats such as N3 and Turtles already fit this criteria. However, this approach limits its audience to those with knowledge of RDF and domain-specific ontologies. And even for sufficiently trained users, writing precise and atomic RDF statements flies in the face of the Wiki’s goal of being “quick”.
A more ambitious approach would be to design a more traditional Wiki-like text format whose structure could be easily represented as RDF. However there are several challenges to creating a mapping to generic RDF or some general purpose ontology for content. First, current Semantic Web standards, such as OWL, are not yet powerful enough to inference equivalencies between a representation in a content ontology and its appropriate domain-specific ontology. Second, the most intuitive markup structure for a particular application doesn’t always submit to a straightforward mapping to RDF. Finally, there’s the practical issue that representing structural elements in free form text as RDF creates a tremendous volume of RDF statements, especially if order is preserved.
Because of these limitations, Rhizome’s approach is to use a Wiki-like text format (dubbed ZML) that is flexible enough to express arbitrary structure but doesn’t specify a particular translation to RDF. Instead, the system determines which translation rules to apply based on the content of the text.

Unlike other Wiki text formats, all structural elements in ZML can be arbitrarily nested (relying on whitespace much like the indentation rules found in the Python programming language) and annotated with attributes. The result of parsing ZML is an XML document and in fact ZML can used as a simple, concise alternative syntax for XML. This design enables the user to easily use microformats[5] or domain-specific XML vocabularies (for example, Rhizome supports vocabularies from the Apache Forrest and Docbook projects). Another advantage is that this lets arbitrary HTML or XML be converted to ZML, enabling round-trip conversions. For example, users can write content in ZML, edit it in a WYSIWYG (X)HTML editor, or process it with specialized tools that consume XML, and then view it as ZML again.
ZML also has syntactic constructions to make it easy to explicitly express semantic distinctions that are elided in other Wiki text formats. For example, we must distinguish between creating a reference to a WikiName (which, in our case, corresponds to a RDF resource name) and creating a hyperlink, which has explicit presentational intent and generally implies a relationship between the content and the link target. Similarly, we must distinguish between anchors and their common use as a way to name document sections.


Sample FAQ - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8000/sampleFAQ?action=edit
View Comments (0) Edit Revisions Metadat
About
Home
Edit
sidebar
to add your links.
Editing sampleFAQ
Title
Upload File:
Browse...
OR edit text here:
Ad hoc structural elements
Unordered list
Hyperlinks
Metadata annotation

Title: Sample FAQ

```
#?zml0.8
<faq:
 <question:
   Where can I find out more about ZML?
 <answer:
   Here are a few links:
   * [ZMLQuickReference]
   * [ZML Home page|http://zml.liminalzone.org].
   [Any more we should add? | todo:revisit;]
```

Fig. 1. A screenshot of a page being edited in the Rhizome Wiki, with aspects of ZML syntax highlighted.

ZML doesn't directly translate into RDF; instead it relies on “shredding”, the process Rhizome uses to bridge implicit and explicit semantics. Shredding is a flexible framework for specifying rules for characterizing semi-structured content with RDF and providing an ontology that can precisely describe the relationship between the source content and the resulting statements.
Rhizome lets users create rules that trigger shredding on the basis of the content's type. For example, shredding an RDF/XML document would consist of parsing the RDF; shredding an (X)HTML document could invoke invoking a GRDDL (Gleaning Resource Descriptions from Dialects of Languages) [6] XSLT stylesheet; and shredding an MP3 file would consist of extracting the metadata out of the embedded ID3 tag. Using RxPath's support for RDF named graphs (see below), Rhizome can retain the relationships between an instance of content and statements extracted from

it, enabling it to know, for example, that the statements might be out of date when content has changed.
Rhizome also lets users directly view and edit raw RDF in ZML via RxML, an alternative syntax to RDF with the goal of enabling novices to read and edit RDF using a metaphor conceptually similar to and only incrementally more complicated than application properties file formats such as Microsoft Windows' .ini files. Although RxML can express any set of RDF statements, it presents the RDF in a constrained, simplified manner: as a list of resource URIs, each of which has a set of property name-value pairs.

2.2 Unified

Providing a unified namespace for users requires a strategy for mapping WikiNames to RDF resource URIs. One simple approach would be to treat the WikiNames themselves as a resource URI, e.g. by introducing a "wiki:" URL scheme. It is obvious that given the decentralized nature of the Semantic Web this approach could not scale without name conflicts arising. Alternatively, we could generate a unique URL from a WikiName; for example by using the actual URL to the web page that corresponds to the WikiName, or by pre-pending some application specific base URI. However, this contradicts the principle of a unified namespace by essentially creating separate namespaces -- users would not be able use to WikiNames to refer to resources outside the system without some way to refer to those namespaces.
Thus Rhizome assumes that in order to provide a single, flat namespace of WikiNames that is universally addressable we need to create a level of indirection between a RDF resource URI and its WikiName, and accept that the determination of this relationship is dependent on the context it appears in. WikiNames are treated as a property of a resource, with only slightly stronger semantics than RDF Schema's "rdfs:label" property. When a WikiName is referenced in content, it is up to the shredding process to assert a relation between it and a RDF resource. This is appropriate because the question of how closely that name should be "bound" to an RDF resource is dependent on the needs of the specific application and what assumptions can be made about the context in which it appears.

2.3 Tolerant

The principle of tolerance is harder to achieve with Semantic Web data than the plain text found in traditional Wikis because Semantic Web data is precise and machine consumable and so very often requires some degree of validation. Rhizome allows an application to maximize the tolerance allowable by providing partial, incremental and ad-hoc of validation of RDF using Schematron. Thus validation can be accomplished without having to use complex ontology languages such as OWL, which can often break down in the face of inconsistency. Schematron[7] is a validation language that uses XPath expressions as assertions about the validitity of a XML document. Using RxPath (described below), Schematron can be used to validate a RDF model. The benefits of using Schematron to validate XML also apply to validating RDF:

Schematron allows complex, ad-hoc assertions to be expressed that can't easily be expressed in other schema languages. For example, because OWL is based open world model, it can't define constraints that apply against the entire model such as uniqueness or default values. And compared to languages like OWL, Schematron is easier to write and understand and requires much less specialized knowledge.

2.4 Open

Like tolerance, it is more difficult to achieve openness in Semantic Web applications than with traditional Wikis. Rhizome attempts to balance openness with security by providing an authorization scheme that is powerful yet unobtrusive (one that doesn't impose an addition work where it is not needed). Rhizome lets the application define authorization rules for the addition and removal of arbitrary RDF statements using the notion of access tokens that guard resources. This conceptually simple model can be used to build fairly complex authorization rules; for example, one that allows a guest account to create a new user account for herself, but not modify or create other accounts or objects. However, the RDF model can make it difficult to create these rules because of the very fine-grained nature of RDF resources (for example, even very simple types objects can require anonymous resource nodes). Rhizome deals with this by allowing the application to declare properties that are used to partition an RDF graph into coarser-grained objects to apply authorization to.[1] Rhizome also maintains a revision history of all changes to the system using named graphs to model transactions. This allows changes to be monitored and inappropriate modifications to be reverted when necessary.

3. Discussion

This section provides an overview of Rhizome's architecture.

3.1 Architecture

Figure 2 illustrates the overall architecture of the Rhizome framework. Components are arranged as a stack in which higher-level components depend on the lower-level components, but not vice versa. Consider each layer from bottom to top:

[1] Not discussed here is ways in which class inferences add complexity when rules based on class types are allowed.

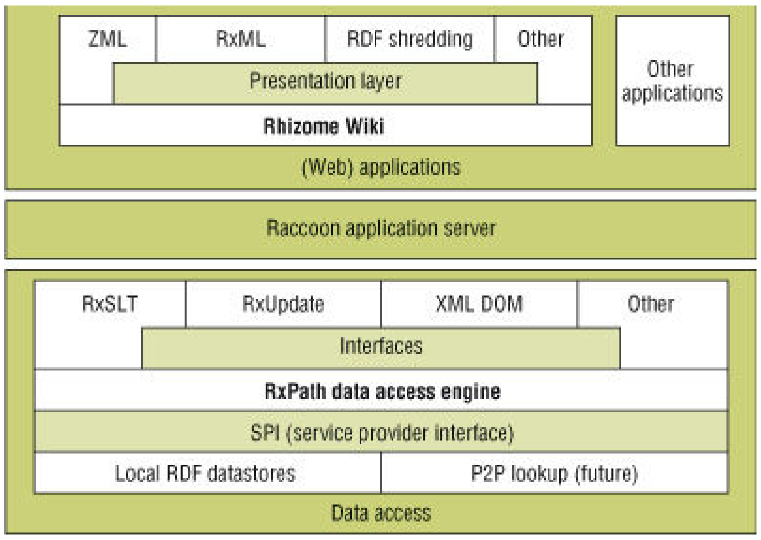


Fig. 2. Rhizome's architecture.

3.1.1 RxPath data access

RxPath is an RDF data access engine that provides a deterministic mapping between the RDF abstract syntax and the XPath data model. This lets users access RDF data stores as a (virtual) XML DOM (document object model) and query them using RxPath, a language syntactically identical to XPath 1.0. This approach allows the full range of XPath-based languages to be used to query and manipulate RDF models -- for example, XSLT for presentation and transformation, XUpdate[8] for modification, Schematron for validation, and XForms for presentation and modification -- without having to make any syntactical changes to those languages.

RxPath maps the set of (subject, predicate, object) triples in an RDF model into a virtual and possibly infinitely recursive tree in which:

- the root has a child node corresponding to each resource in the model,
- each resource node has child nodes for each statement that it is the subject of
- each statement node has a single child node corresponding to the statement's object.

If the statement's object is a resource, it might in turn have child nodes that correspond to the statements that the resource is subject of, and so on. Given such a tree, an XPath expression such as `/foaf:Document/dc:creator/*` will select a set containing all the authors of each document resource in the RDF model.
RxPath also supports "named graphs"[9] (also known as contexts), a common extension to the RDF model that is used to partition RDF statements into groups. RxPath uses a unique approach to contexts by treating them not as a one-to-one mapping with a subgraph of an RDF model, but as a collection of subgraphs composed through union and difference operators. This enables Rhizome to use

contexts simultaneously and efficiently to model many different concepts, such as metadata versioning, transactions, provenance, application partitioning, and personalization (user customizations). For example, Raccoon's transaction log of changes made to the RDF store is represented as a collection of contexts, each of which adds or subtracts from the previous context. Using contexts lets Rhizome capture when, where, how, and by whom a set of statements was made.

3.1.2 Raccoon application server

Raccoon is a simple application server that uses an RDF model for its data store. Raccoon uses RxPath to translate arbitrary requests — such as HTTP requests or command line arguments — to RDF resources. Each of these can be associated with style sheets in RxSLT and RxUpdate languages, which can generate responses or update the RDF data store.

Raccoon's goal is to present a uniform and purely semantic environment for applications. This enables the creation of applications that are easily migrated and distributed and that are resistant to change. Raccoon is designed primarily for applications that look at the world as a universe of RDF statements, but it also works with XML-centric applications. Raccoon isn't designed to be a full-featured application server and in fact will often be embedded in another application server. Raccoon's job as an application server is a narrow one—to map a request to a response, possibly modifying the state of the application in the process:

Request $\rightarrow$ Application (Rules + Store) $\rightarrow$ Response

A request is a dictionary of simple values, and an application defines a pipeline of RxPath expressions that transform the request into the response. Raccoon presents both the request and the application's state using the RxPath data model. This approach enables the creation of applications that can be transparently distributed and aggressively cached. Application code is always executed within the context of a request. There are external requests, such as HTTP requests, and internal ones, such as the requests sent when an application starts or stops. Raccoon also provides basic transaction coordination for managing updates to the RDF store. Using contexts enables the application to choose an appropriate consistency model for its needs. If full global atomic consistency isn't needed, Raccoon can cache request responses even more aggressively and still provide the appropriate levels of cache coherency.

3.1.3 Rhizome Wiki

Running on top of Raccoon is the actual Wiki application, which offers all the basic functionality found in Wikis, such as letting users create and edit pages on an ad hoc basis; along with some more advanced content management features such as roles and groups, release workflow, and basic facet navigation. Almost all of the Wiki's functionality is implemented in its dynamic pages, which are written in RxSLT, XSLT, and RxUpdate. Users can edit these like any other pages, making it easy to incrementally add and change functionality. They can also use RxUpdate to modify the underlying schema at run-time.This flexibility makes access control very important—to this end, Rhizome uses a flexible schema for authorizing both application-level actions and statement-level changes to the RDF store based on the authorization mechanism described in the previous section.

3.2 Conclusion

This paper has examined some of Rhizome's approaches to applying Wiki design principles to the Semantic Web. Despite the challenges of marrying two very different approaches to capturing knowledge, doing so can help reduce the barriers that often hinder the adoption of Semantic Web technologies, such as high learning curves for users, demands for precision and consistency, and the need to develop domain-specific user interfaces.

References

1. Cunningham, W, Leuf, Bo.: The Wiki Way: Collaboration and Sharing on the Internet Addison-Wesley Professional (2001)
2. Cunningham, W, et. al. http://c2.com/cgi/wiki?WikiDesignPrinciples
3. http://www.dfki.de/~paulb/ABCDEF/ABCDEF.htm
4. Souzis, A: Building a Semantic Wiki. IEEE Intelligent Systems (Sep/Oct 2005) 87-91
5. http://www.microformats.org
6. Hazael-Massieux, D., Connolly D.: Gleaning Resource Descriptions from Dialects of Languages (GRDDL), World Wide Web Consortium (W3C) Note (2005)
7. ISO/IEC 19757-3 Document Schema Definition Languages: Part 3 — Rule-based validation — Schematron (2004)
8. Laux, A., Martin, L.: XUpdate—XML Update Language, XUpdate Working Group Specification (2000).
9. Carroll, J. et al.: Named Graphs, Provenance and Trust, In: Proc. 14th Int'l Conf. World Wide Web (WWW 05), ACM Press, (2005), 613–622.