IEEE Intelligent Systems

www.computer.org/intelligent

Building a Semantic Wiki

Adam Souzis

Vol. 20, No. 5
September/October 2005

Editor: Steffen Staab
University of Koblenz-Landau
staab@uni-koblenz.de

Building a Semantic Wiki

Adam Souzis, *Liminal Systems*

The Semantic Web vision of a "unifying logical language that enables concepts to be progressively linked into a universal Web" (as Tim Berners-Lee put it)[1] is part of a long lineage of dreams of a universal repository of ideas: from Diderot's universal encyclopedia in the 18th century to Vannevar Bush's Memex at the beginning of the computer age to Ted Nelson's Xanadu in the 1970s. However, the Semantic Web's development so far has focused primarily on metadata and carefully designed data structures. To realize Berners-Lee's vision, the Semantic Web must capture and represent content created every day by people without special training—such content includes blogs, emails, and discussion groups.

Rhizome (www.liminalzone.org; download at http://sourceforge.net/projects/rx4rdf) is an experimental, open source content management framework I've created that can capture and represent informal, human-authored content in a semantically rich manner. Rhizome aims to help bring about a new kind of commons—one of ideas. This commons wouldn't comprise just a web of interlinked pages of content, as is the current World Wide Web, but a web of relationships between the underlying ideas and distinctions that the content implies: a permanent, universally accessible interlinking of content based on imputed semantics such as concepts, definitions, or structured argumentation.

The challenges

Semantic Web technologies such as RDF provide a good foundation for creating this kind of network. RDF's two most important characteristics are that RDF resources are globally unique and therefore can be referenced universally, and that the monotonic semantic model doesn't make the closed-world assumption.[2] More specifically, one can always make new statements about a resource, enabling a decentralized network in which to safely discover new facts without invalidating previous conclusions.

However, current Semantic Web technologies fall short of meeting some of the challenges that informal, human-authored content presents. Rhizome focuses on addressing two of these challenges. The first is the ability to handle ambiguity, inconsistency, and multiple views and perspectives. Related to this, we need ways to model content so that we can maintain appropriate semantics even as the content is replicated and altered. The second is Semantic Web technology's ease of use. We need to make this technology dramatically easier to use, for both developers and end users. Semantic Web technologies are difficult conceptually, even for experienced software developers. It's difficult to imagine nontechnical end users effectively authoring RDF statements without software assistance. Even for sufficiently trained users, writing the required precise statements takes much more time and effort than writing informal text. Thus, these two challenges are interrelated, because the less we need precise, consistent statements, the easier it is to create semantic content.

The Rhizome application stack lets developers build Web applications that use familiar technology such as XML and XSLT (extensible stylesheet language transformations) but run in an environment where the outside world (for example, data sources and incoming requests) is presented as a universe of RDF statements. Rhizome is designed for applications such as

- a wiki-like application that lets the user create arbitrary RDF resources and the application logic for presenting and interacting with them,
- a personal note management and publishing tool for keeping track of and extracting metadata and user annotations from a variety of local sources such as plain-text files and comments in source code, and
- a discussion forum in which users can classify and structure responses with typed annotations instead of email-like quoting.

Overview

Rhizome consists of a stack of components (see figure 1). The "Architecture" section describes these components in more detail, but first I'll start with an overview.

1541-1672/05/$20.00 © 2005 IEEE
Published by the IEEE Computer Society

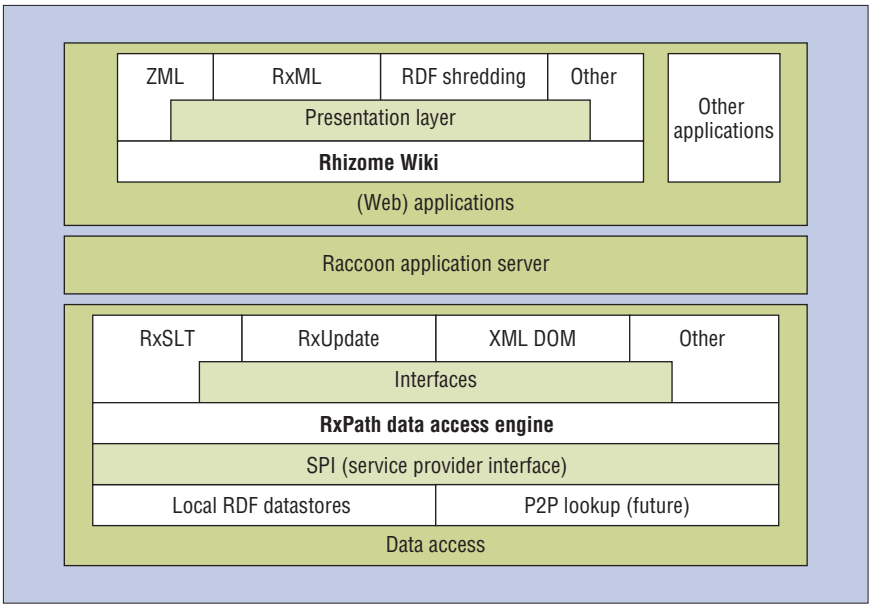


Figure 1. The Rhizome architecture stack.

The components

At the top of the stack is *Rhizome Wiki*, a wiki-like content management and delivery system that treats all content, metadata, and structure as RDF and lets users edit any RDF resource as they would a wiki page. Rhizome Wiki uses a couple of custom text formats to make writing semantic content easier:

- *ZML* is a plain-text formatting language similar to those found in wikis, except that users can create ad hoc structure and use special formatting conventions to indicate semantic intent.
- *RxML* is a simple, alternative format for RDF that aims to be as easy to use as an application properties file. Thus, novices can author and edit RDF metadata. It's specified as XML but designed for authoring in ZML.

Rhizome Wiki runs on top of *Raccoon*, a simple application server that uses an RDF model for its data store. Raccoon uses *RxPath* to translate arbitrary requests—currently HTTP, XML-RPC, and command line arguments—to RDF resources. Each of these can be associated with style sheets in RxSLT and RxUpdate,[3] languages that are subsets of XSLT and XUpdate.

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. Building on this are RxSLT and RxUpdate, languages for transforming and updating the RDF data store; they are syntactically identical to XSLT and XUpdate, respectively. Other XML technologies that rely on XPath can be easily adopted, such as using Schematron to validate an RDF model.

Rhizome in action

To see how Rhizome uses these various components, let's take a look at how a user creates a page of content using Rhizome Wiki. Figure 2 shows three different views of a page created in Rhizome Wiki, illustrating how to create structured content that's transformed into and stored as RDF. The page represents a FAQ (frequently asked questions), a familiar document convention. Rhizome makes it easy to turn relatively informal document conventions like FAQs into semantically meaningful resources—in this case, an RDF resource for each question-and-answer pair. This feature helps fulfill the Semantic Web's promise: If FAQs were exposed on the Web this way, building applications that could aggregate them and provide specialized interfaces for tasks such as browsing, tagging, and rating would be much easier.

Figure 2a shows the page's presentation view, a FAQ with just one question and answer. Notice that the presentation is specific to FAQ resources: When users request a FAQ resource, Rhizome invokes an RxSLT style sheet to transform the RDF resource into XHTML. Because RxSLT is syntactically identical to XSLT, I was able to adopt, with minimal modifications, an XSLT style sheet for displaying FAQs from an unrelated project (Apache Forest, http://forrest.apache.org). This ability to easily adapt XML technology is one way Rhizome strives to make RDF easier to adopt.

Figure 2b shows the same page in edit mode, revealing that the source isn't RDF but ZML. Compared to similar wiki text formats, ZML has some unique features: It lets users create ad hoc structural elements mapped to XML elements or RDF resources and has syntactic constructs for explicit semantics such as the metadata annotation in figure 2b. When a user saves a page of ZML content, Rhizome Wiki converts the ZML to RDF before saving the RDF in its data store. We can see this RDF in figure 2c, which shows the sample page's metadata view. The label "metadata" is a bit misleading because Rhizome stores everything as RDF, so, in a sense this is also the data view. This view displays the RDF using RxML.

The RDF that makes up this page comes from three sources: the process of converting the ZML to RDF, called *shredding*; the structural properties created by the handler that handles the "save" action when an edit is completed; and the user-defined metadata. From this view, users can edit any of this RDF directly, thus enabling them to change the wiki application's behavior and structure. To mitigate the inherent dangers of this level of openness, Rhizome Wiki provides fine-grain authorization and validation alongside the use of contexts.

Architecture

As figure 1 illustrates, Rhizome's components are arranged as a stack in which higher-level components depend on the lower-level components, but not vice versa. We can see how the components are integrated by examining each one from the bottom to the top of the stack.

RxPath data access

The main motivation for creating RxPath was to make RDF easier to use, especially for developers more familiar with XML than RDF. RxPath does this in two ways. First, it enables users to apply familiar XML technologies to RDF models in a straight-

forward way. For example, users can extract and format content in an RDF model using a standard XSLT style sheet without relying on any extension functions or elements. Users can author and even test and debug the style sheet with standard XSLT development tools. Second, RxPath lets us visualize an RDF model much like a standard XML DOM, reducing much of the conceptual impedance between RDF and XML. This should reduce RDF's learning curve for a developer with competence in XML and related technologies and therefore contribute to RDF's widespread adoption.

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's the subject of, and
- each statement node has a single child node corresponding to the statement's object.

If the 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[4] (also known as contexts), a common extension to the RDF model 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.

Raccoon application server

Raccoon's goal is to present a uniform and purely semantic environment for applications. This enables the creation of applica-

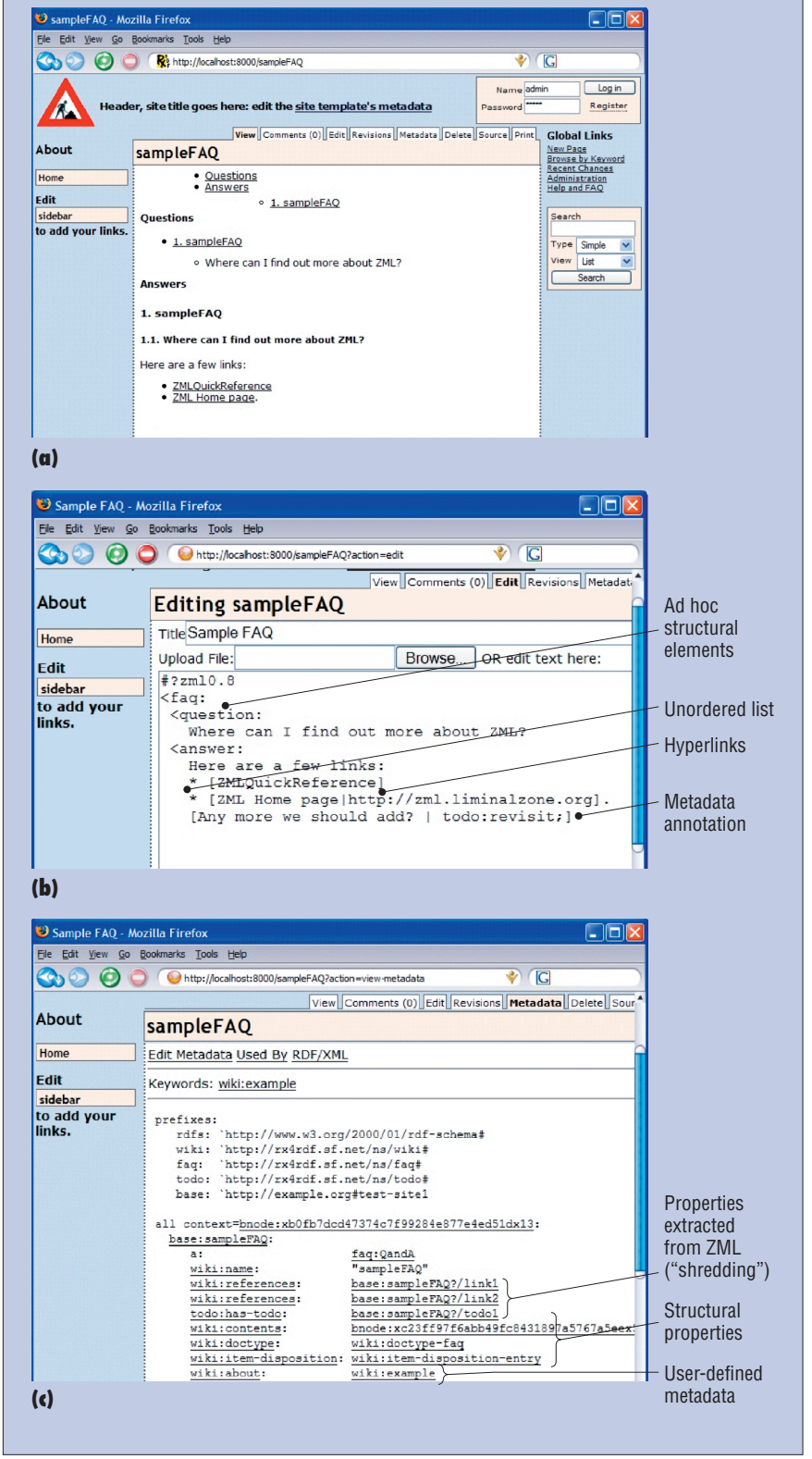


Figure 2. (a) A sample page created in a fresh installation of Rhizome Wiki. (b) A screenshot of Rhizome Wiki in edit mode on a page created using ZML. (c) The metadata view of the sample page showing the underlying RDF.

tions 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 → Application (Rules + Store) → 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.

To get a sense of how an application built on Raccoon uses rules to implement functionality, take a look at the metadata in figure 2c. In particular, take note of the following properties:

- *wiki:name*. When Raccoon receives a request for a URL, it uses this subproperty of rdfs:label to find the RDF resource that corresponds to the URL. The rule is written in RxPath as `/*[wiki:name=$_path]`.
- *wiki:doctype*. After the previous rule selects the resource, other rules look for a display handler on the basis of the type of resource and the requested action. The rule that invokes the RxSLT resource that transforms the FAQ resource into HTML is `/*[wiki:handle-doctype= $__resource/wiki:doctype]`.

Each rule I've discussed is defined in the application configuration file for Rhizome Wiki. By simply adding rules and corresponding metadata, we can add sophisticated functionality. Many Rhizome Wiki features, such as its release workflow and authorization model, are implemented in this manner.

ZML and RxML

A common assumption made about the Semantic Web is that special software will assist in authoring content for the Semantic Web. But until the day this functionality is built into word processors, text editors, Web page forms, and all the other places a user can enter text, this assumption will limit the Semantic Web's growth. Instead, Rhizome follows wiki design principles[5] and introduces ZML, a plain-text format with simple text conventions to guide presentation, thus allowing users to create content with explicit semantics anywhere that text can be entered. Another advantage of using a wiki-like text format for authoring semantic content is that, unlike word processor formats or even HTML with CSS, wiki text formats provide very limited presentational options. This limitation makes it hard for naive or undisciplined users to express semantics through presentation (for example, by using text effects).

> Rhizome uses a novel architecture to enable a unique approach for addressing some of the challenges of creating semantically explicit content, but many more challenges exist.

However, enabling the authoring of content with explicit semantics requires significant enhancements to existing wiki formats. First, users must be able to create arbitrary structures. To this end, users can use ZML as a simple, concise alternative syntax for XML, enabling them to author any XML construct. In fact, the result of parsing ZML is XML.

One advantage of this approach is that it lets arbitrary HTML or XML be converted to ZML again, enabling roundtrip conversions. For example, users can write content in ZML, edit it in a WYSIWYG HTML editor, or process it by some specialized tools that consume XML, and then view it as ZML again.

On the other hand, ZML provides no intrinsic translation to RDF—instead, Rhizome Wiki maps the resulting XML using rule-based shredding, which I'll describe in a moment. Several reasons for this exist. First, current standardized ontology languages, such as OWL, aren't yet powerful enough to infer equivalencies between a generic RDF representation 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; shredding gives us flexibility to handle these cases.

Finally, there's the pragmatic issue: Always representing all structural elements as RDF creates a tremendous volume of RDF statements, especially if order is preserved. Creating this mass of RDF statements would also open heretofore unexplored (by Rhizome) questions of composition: how to reason about, present, modify, and reintegrate resources that are parts of a larger resource.

ZML needs special syntax to make it easy to explicitly express semantic intent. One requirement is the ability to create metadata annotations of both content and structural elements (for example, section headers or list items). We also need formatting rules that can express semantic distinctions that are elided in current wiki text formats. For example, we must distinguish between creating a reference to a WikiName (which, in the case of Rhizome Wiki, is an RDF resource's 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.

Figure 2b includes an example of ZML that illustrates some of these features, demonstrating the syntax for creating structural elements (transformed to XML elements), unordered lists, hyperlinks, and metadata annotation.

Rhizome Wiki also uses RxML (see figure 2c). Although RxML can express any set of RDF statements, it presents the RDF in a constrained, simplified manner: a list of resource URIs, each of which has a set of property name-value pairs. Its goal is to let novices read and edit RDF metadata

using a structure conceptually similar to and only incrementally more complicated than application properties' file formats such as Microsoft Windows' .ini files.

Rhizome Wiki

Rhizome Wiki offers all the basic wiki functionality, 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 Rhizome'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 migrate the underlying schema at runtime. 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.

One factor limiting the Semantic Web's growth is the reliance on generic formats for representing RDF. Although RxML tries to mitigate this (especially compared with the complexity of the W3C's standard XML representation of RDF), it's unrealistic to expect the vast installed base of applications to be enhanced to support RDF anytime soon. Rhizome's approach to this issue is shredding, or providing a framework for extracting RDF statements from a variety of content formats.

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 HTML document with a GRDDL (Gleaning Resource Descriptions from Dialects of Languages)[6] link would consist of invoking the referenced XSLT script; and shredding an MP3 file would consist of extracting the metadata out of the embedded ID3 tag. Using contexts, 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.

For instance, when the user saves the ZML in figure 2b, shredding occurs, resulting in the RDF in figure 2c. A shredder can be associated with content types, and shredding is recursive. So, in this case, the first shredder invoked was for ZML, and it simply parsed the ZML to XML and invoked the XML shredder. The generic XML shredder is an XSLT style sheet that recursively invokes shredders specific to an XML namespace. Finally, the shredder associated with the FAQ XML namespace is an RxUpdate script that for each FAQ element adds an RDF resource to the store on the basis of a specific FAQ ontology.

Peer-to-peer future

A "commons" implies shared resources, equally available to all. If we think concretely about how to build the commons of ideas I mentioned in the introduction, we realize it must be built on a robust, decentralized, peer-to-peer infrastructure. One possible architecture could look like this:

- All (public) RDF statements are stored in a global distributed store and are accessed through the RxPath processor through next-generation, distributed hash tables such as PGrid.[7]
- Application files are stored in a peer-to-peer content distribution network such as Coral.[8] Unlike a typical CDN, the nodes would have Raccoon-like processors that would actively process the content using the global RDF store. During processing, the cached results would also be added to the CDN.

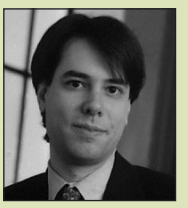
Several of Rhizome's design elements support this type of architecture:

- Raccoon presents a purely semantic view of the environment that enables applications to be independent of their physical environment.
- Raccoon's rules-based approach makes it easer to distribute and cache application processing.
- RxPath's path-based query model can use peer-to-peer lookup primitives more easily than tuple-based query languages such as SQL (Structured Query Language) or SPARQL (Query Language for RDF).
- RxPath's use of contexts lets applications choose a consistency model appropriate for a decentralized, distributed environment (contexts also enable statements to be clustered for efficient distributed querying).

Rhizome's implementation is freely available, open source, and implemented in Python. Although it's fully functional, it's still very much in an experimental state. Rhizome uses a novel architecture to enable a unique approach for addressing some of the challenges of creating semantically explicit content, but many more challenges exist. Some of the more difficult ones include the stability of names, the composition of resources, change management, and conflict resolution. These challenges will shape Rhizome's future development. ■

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.,* May 2001, pp. 28–37.
2. P. Hayes, *RDF Semantics*, World Wide Web Consortium (W3C) recommendation, Feb. 2004; www.w3.org/TR/rdf-mt.
3. A. Laux and L. Martin, *XUpdate—XML Update Language*, XUpdate Working Group specification, 14 Sept. 2000; http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html.
4. J. Carroll et al., "Named Graphs, Provenance and Trust," *Proc. 14th Int'l Conf. World Wide Web* (WWW 05), ACM Press, 2005, pp. 613–622.
5. W. Cunningham, *Wiki Design Principles*, http://c2.com/cgi/wiki?WikiDesignPrinciples.
6. D. Hazael-Massieux and D. Connolly, *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)*, World Wide Web Consortium (W3C) note, 13 Apr. 2004, www.w3.org/TR/2004/NOTE-grddl-20040413.
7. K. Aberer et al., "GridVine: Building Internet-Scale Semantic Overlay Networks," *Proc. 3rd Int'l Semantic Web Conf.* (ISWC 04), LNCS 3298, Springer, 2004, pp. 107–121.
8. M.J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing Content Publication with Coral," *Proc. 1st Symp. Networked Systems Design and Implementation* (NSDI 04), Usenix Assoc., 2004, pp. 239–252.



Adam Souzis is founder of Liminal Systems. Contact him at asouzis@users.sourceforge.net.