Building Interoperable Metadata on the Web

Hal Abelson Ben Adida (Eric Miller?) (Stefano Mazzocchi?)

January 6, 2006

Abstract

1 Introduction

Data interoperability is the soul of the web. Interoperable data makes it easy for people to accumulate information and build on each other's work through collaborations that are widely distributed and loosely coordinated. Interoperable data makes it easy for implementors to create tools that operate across heterogeneous systems, without requiring close coordination between tool builders and data providers. Interoperable data makes it easy for Web users to copy and paste material from one page to another and to reuse information in different contexts. Data interoperability, and the basic archiecture of universal identifers (URIs) and common formats (HTML) is the critical enabler of the network effects that make the Net the Net.

What's true for data is true for metadata. Metadata interoperability becomes increasingly critical as Web becomes increasingly a Web of data navigated with the help of tools and services, rather than only a Web of documents browsed manually. The opportunity for interoperability becomes apparent with the emergence of application Web sites designed to help individuals manage, categorize, and explore data: Delicious [??] manages bookmarks, Flickr [??] manages photos, Blogger [??] and Typepad [??] manage news and commentary, and Gmail [??], Yahoo [??], and MSN [??] manage email. These sites have developed finely tuned methods for annotating and navigating the specific type of data they manage. One popular trend is simple keyword tagging, also known as ¡em¿folksonomies¡/em¿, where users assign multiple single-word tags to data elements, either scoped individually as in Gmail or communally as in Flickr. [1]

Yet even simple tagging is implemented differently by different systems, and consequently individual applications are lacking in interoperability: a tool builder who wants to extract structured metadata from any one of these applications must be intimately aware of its data schema and often its customized API.

To date, the only widely successful example of web-based metadata interoperability is RSS, a single XML data schema built for the linear publication of news items. Based on RSS, meta-applications like Technorati [??], Bloglines [??], and Google and Yahoo's customized homepages

[1] The term "folksonomy" is credited to Thomas Vander Wal, who describes it in his blog as "the result of personal free tagging of information and objects (anything with a URL) for one's own retrival." (See http://www.vanderwal.net/random/category.php?cat=153], visited Dec. 31, 2005.) Interestingly, the same blog page notes that the that academic paper cite the definition of the term in the Wikipedia entry, and yet that definition continues to change as the Wikipedia entry is updated. This illustrates the need for ways to address data provenance on the Web. Provenance is another key issue for metadata, but one that is outside the scope of this paper.

provide some amount of automated, cross-site data aggregation and filtering. Some more adventurous applications, like Technorati's tag search, or the various Google-Maps-based "mash-ups", use customized APIs offered by content communities to enable features more advanced than the opaque aggregation of RSS. Unfortunately, it is unclear how Technorati or these innovative mash-ups might handle more than a handful of data sources, because their code is highly customized according to the sites they consume.

Ideally, interoperability on the web would let tool builders consume data emanating from numerous applications with which they have no privileged relationship. For most applications, there would be no need for custom APIs, and adding a new data source to a meta-application would require no code change. Such interoperable metadata would democratize the current mash-up trend, a stimulate remixing and customization of the ever-growing mountain of Web data into highly-specialized and highly-useful applications.

This paper explores steps that web publishers can take to help fulfill this vision of an interoperable web of metadata. We consider nascent metadata discovery techniques and describe the advantages and disadvantages of various options. We posit that guidelines for web publishers should be clear and manageable without interfering in each community's internal needs and features. The questions are simple: what should a web publisher do to facilitate the tool maker's job? How should individual communities publish data so that they can be easily remixed, both manually and in automated ways? How do a Web publishers act as good citizens of an interopable metadata world?

2 This Paper

While the questions may be simple, the answers require balancing multiple, often conflicting requirements. Web publishers want good support for their user communities; users want compelling, consistent experiences across different sites. Tool builders want well-known public interfaces for accessing sites programmatically. Everyone wants to enable collaboration, remixing, and evolution without the need for pre-arrangement on tight coordination, and no one wants existing sites and services to stop working.

To help assess how proposed architectural approaches measure up against these requirements, we'll consider some fictional participants in a world of Web metadata:

Paul is a web publisher. His site, Blogr, manages simple text blogs. Peter is another web publisher whose site, Shutr, manages photo blogs, and Patrick is yet another web publisher whose site, Podr, manages podcasts. Ursula uses all three sites: Blogr for her thoughts on American Cuisine, Shutr for photos of the various restaurant meals she eats, and Podr for her weekly recipe podcast.

Meanwhile, Tim and Theodore are toolbuilders. Tim manages a site called Blogerati that aggregates user content according to various categorization schemes. Blogerati does not let users upload content directly: all content is consumed from other sites like Blogr, Shutr, and Podr. Theodore builds a desktop news aggregator called DeskBlog.

We first consider what an ideal metadata world might look like. What should Paul, Peter, Patrick, Tim, Theodore, and Ursula be able to do? What kinds of resources would they need to participate? We take into account that an ideal world cannot simply come into existence at once: it must include transition paths for existing published sites. We then consider how various current data/metadata schemes perform against this ideal: XML or microformats available over

HTTP GET, and custom APIs based on Representational State Transfer (REST) [**?**]. We assert that accessing data using HTTP GET is preferable to using custom APIs: one should try to mimic the success of RSS in this respect. Going further, We find that the Web Consortium's Resource Definition Framework (RDF) [**?**] is a good mechanism for expressing generic metadata. RDF is usable without any centrally defined taxonomies and the "flatness" and universally-identified aspect of RDF properties promotes easy interoperability.

We then explore how current formalisms, including XML and XHTML, as well as microformats, effectively define small-scale worlds of metadata, where machines can interpret one another. Such architectures are useful, but insufficient: they fail to achieve large-scale interoperability because no data in them interpretable without knowing the specific schema, even if schemas can share components.

We conclude that RDF is the only currently existing framework that enables a large-scale world of interoperable metadata. We show how existing techniques can be "upgraded" to RDF using the "Gleaning Resource Descriptions from Dialects of Languages" technique (GRDDL) [**?**]., Ww then consider RDF/A [**?**] a recently developed serialization mechanism for RDF that lends itself particularly well to Web publishing. We explore how RDF/A enables the primary browsing interface, XHTML, to also be the metadata interface. RDF/A is also appropriate when a Web publisher does not wish to become the choke-point for new metadata definitions.

Finally, we describe how these architectural frameworks play out in some existing real applications with existing significant user communities. **finish this to talk about CC and whatever other example we're going to use**.

3 Principles for an Ideal Metadata World

In an ideal world, web publishers and tool builders enjoy seamless data interchange assisted by interoperable metadata. Publishers Paul, Peter, and Patrick need only perform minimal work to prepare their data for universal consumption. Toolbuilders Tim and Theodore can immediately and automatically extract metadata from any publisher. This ideal metadata world stands on five pillars of metadata design, which we outline below. Not all of these are completely compatible: a real-world solution will likely have to strike a compromise between them.

3.1 Publisher Independence

Paul manages the Blogr user community, where Ursula blogs about the latest developments in American Cuisine. Blogr lets users like Ursula add tags to their posts. Meanwhile, Peter allows users of Shutr, where Ursula uploads photos of restaurant meals, to add information about the camera used to take each posted photo and the geographic coordinates of the photo. Paul and Peter expect to continue to define such features without interference from external standards-bodies or from one another. This is the principle of **Publisher Independence**. A web publisher should be able to define the details of his metadata schema with minimal constraints from external sources.

3.2 Data Reuse

Ursula uses Theodore's desktop news reader to stay up-to-date on various blogs. She also uses Tim's content aggregator site to find new blogs, and sometimes she browses Blogr via her normal

web browser to discover new and interesting recipes. Paul, the publisher of Blogr, would prefer to use only one data format to feed into each of these different uses, whether for human-readable rendering or machine-readable parsing and aggregation. This is the principle of **Data Reuse**. A web publisher should be able to attach metadata to existing, rendered-for-humans data, without duplicating the actual data content.

3.3 Metadata Self-Containment

In her Blogr postings on American Cuisine, Ursula often copies and pastes segments from other blog entries (properly licensed for such reuse, of course) to comment on the proposed recipe and provide her own insights. She expects her expert toolbuilder friend, Theodore, to build her a blogging client that will automatically carry along any metadata from the quoted blog post when she copies and pastes. This is the principle of **Metadata Self-Containment**. Metadata should be closely bundled with the data to which it pertains. [**MORE HERE?**].

3.4 Schema Modularity

Tim, the manager of Blogerati, wants to aggregate as many blogging sites as possible, including Blogr, Shutr, and Podr. Whenever possible, he indexes specific metadata fields, e.g. title, posting date, and folksonomy tags. If some of this metadata is absent, he would still like to index the rest, while ignoring the metadata he doesn't care about (yet). This is the principle of **schema modularity**. Consumers of web metadata should be able to recognize individual metadata components within larger, partially unrecognized larger metadata sets.

3.5 Constructive Evolvability

As Peter, Paul, and Patrick expand their individual web applications, they may discover that their metadata concepts map fairly tightly to other, existing concepts. They may wish to define equivalences between their local metadata concepts and other, established metadata terms. Alternatively, another web site may wish to declare such attribute equivalences. Tim should be able to take into account any source of such metadata equivalence. Over time, as Tim chooses his data sources, more detailed relationships between various consumed sites will emerge. This is the principle of **constructive evolvability**. Metadata concepts should be relatatable and extensible by anyone, not just the original publishers.

4 Survey of Metadata Solutions

Data Formats:

- parallel XML (RSS)
- parallel RDF/XML
- GRDDL'ized parallel XML
- GRDDL'ized HTML
- microformats

- RDF/A

[[**WORK HERE**]]

How should Paul, the owner of Shutr, publish structured metadata such that programs built by Tim, the toolbuilder, might be able to automatically download and parse this metadata? We consider, for now, the case where Tim is writing a Shutr-specific screensaver that displays randomly-downloaded images from Shutr. We consider the two important aspects of metadata sharing: discovery and parsing.

4.1 Parsing: Syntax of the Metadata

Paul will likely develop some abstract schema for his photo metadata based on the specific information he stores on the server: camera details, exposure settings, human annotations of the photos, etc... He might then wonder how to publish this data for public consumption. Clearly, a platform-independent mechanism is required. In addition, the syntax should support relatively complex schema structures so that Paul's extensive photo metadata can be expressed fully. A language like XML is a clear fit for this requirement.

4.2 Discovery: Obtaining the Metadata

Before Tim can start parsing the structured Shutr metdata, he needs to know ¡em¿where¡/em¿ to find it. In particular, how is each photo named, and using what protocol can one access the photo's metadata? As we are discussing web-based metadata, we assume underlying HTTP is available to all parties.

Should Paul layer additional structure on top of HTTP, e.g. REST or SOAP, the way Flickr does? In fact, for exporting web metadata, this is not necessary. Flickr requires an additional layer because the basic HTTP protocol – effectively the ¡tt¿GET¡/tt¿ and ¡tt¿POST¡/tt¿ methods – cannot fully describe an API used for structured data updates, e.g. photo uploading. For ¡em¿reading¡/em¿ web-associated metadata, however, there is already a well-established resource identification mechanism – URIs –, and a perfectly functional mechanism for accessing it: the HTTP ¡tt¿GET¡/tt¿ method. Human browsing of a web site requires only the ¡tt¿GET¡/tt¿ method (the ¡tt¿POST¡/tt¿ method is used for data updates, not browsing). Browsing a web site's metadata is functionally equivalent, and can be done the same way.

4.2.1 Associating Metadata With Data

If Paul publishes photos at URIs like ¡code¿http://photoworld.net/photos/12345¡/code¿, then the metadata for such a photo may simply be retrieved at the URI ¡code¿http://photoworld.net/metadata/photos/1234
The association between the two can be indicated by the XHTML for the original resource using a ¡tt¿LINK¡/tt¿ tag:

[[example of LINK REL="META"]]

Alternatively, Paul might want to express structured metadata as part of the original XHTML document he delivers. This is particularly important to help keep the data and its metadata tied together through multiple redistributions. ¡b¿IMPORTANT¡/b¿.

[[MORE HERE]]

4.2.2 End-to-End for Metadata Publishing

The most important feature of such a simple, data-centric approach to metadata is that it provides the ultimate flexibility in metadata production and consumption. With simple HTTP access to the metadata, Paul can begin to publish metadata without complex API support like SOAP or XML-RPC: static, HTTP-served content is enough. Alternatively, Paul can use as intricate a publishing environment as he chooses On the consumption front, Tim needs nothing more than a simple HTTP ¡tt¿GET¡/tt¿ to obtain the metadata. Alternatively, he can layer more complex APIs that encompass multiple HTTP ¡tt¿GET¡/tt¿s and result filtering to accomplish more intricate metadata querying.

By keeping the metadata transport as simple as can be, web publishers of all kinds can contribute, and metadata consumers of all kinds can participate.

4.3 Examples: XML and MicroFormats

4.3.1 XML

XML is a dominant choice for cross-platform, metadata expression. A web publisher might use XML as a parallel mechanism for expressing metadata, where the primary browsing interface is separately served as HTML. For example, Paul's photo site might present the following HTML, at URL ¡tt¿/myphotos.html¡/tt¿:

¡div class="codeblock"¿¡!–#include virtual="paul-photos-html.html.htmlsafe" –¿¡/div¿

In parallel, Paul might offer the URL ¡tt¿/myphotos.xml¡/tt¿ (with automatic client detection so that ¡tt¿/myphotos¡/tt¿ serves HTML or XML interchangeably):

¡div class="codeblock"¿¡!–#include virtual="paul-photos-xml.xml.htmlsafe" –¿¡/div¿

This approach is insufficient in two major ways: ¡b¿metadata maintenance¡/b¿ and ¡b¿interoperability¡/b¿.

For maintenance, note that the metadata and the primary viewing interface are distinct. While these two could originate from the same database row, a user cannot easily correlate the human-readable and machine-readable versions without significant effort. Cutting and pasting data with its machine-readable counterpart is non-trivial.

In terms of interoperability, note that the specific XML schemas is required to determine that each ¡tt¿photo¡/tt¿ is, in fact, its own resource with a set of attributes.

[[MORE HERE...]]

4.3.2 Microformats

Microformats are "designed for humans first." They use HTML constructs to add machine-readability to existing HTML, with well-defined schemas referenced in the HTML ¡tt¿profile¡/tt¿ attribute. With these well-defined schemas and local-only attributes, microformats are effectively a way to combine the XML machine-readable data within the HTML human-readable data.

For example, the above photo information could be expressed in a single HTML document:

¡div class="codeblock"¿¡!–#include virtual="paul-photos-microformat.html.htmlsafe" –¿¡/div¿

Clearly, microformats solve the issue of metadata maintenance and, mostly, portability (one has to be careful to copy the ¡tt¿profile¡/tt¿ information along with any included metadata). However, microformats, like XML, do not have truly modular attributes. Shared attributes across schemas cannot be extracted without intimate knowledge of the enclosing schema.

5 Folksonomies and RSS: Examples of Successful Metadata

In order to devise an approach to interoperable metadata, one should understand the details of current, successful approaches to metadata expression, discovery, and consumption. We consider two examples here: folksonomies and RSS, each of which succeeds in its own, individual way. We will show how the combination of their advantages brings us much closer to our metadata utopia.

5.1 Folksonomies

A number of web applications – e.g. delicious, Flickr, gmail – implement Folksonomies, also known as tagsonomies or simply tagging. Users can apply various tags to their data and later retrieve their data – or data provided by others – by following the trail of tags. Tags are usually simple, often even single words. Tags eschew hierarchy: precision is obtained from a combination of tags, not from inherently specific single categories. In some cases, the meaning of a tag is specific to a single user (gmail), in other cases, to the whole community (Flickr). Well-documented and highly-successful feedback mechanisms are used to enable the consistent establishment of meaning for each tag.

Almost every site that uses tags also provides some kind of API to help programs access the data according to the tags in question. For example, Flickr defines the REST API call ¡tt¿flickr.photos.search¡/tt¿ with includes the argument ¡tt¿tags¡/tt¿, a set of tags to search for. The return value is an XML document that contains references to photos tagged accordingly. Similarly, Delicious offers a number of APIs to access a user's bookmarks, including also a REST API ¡tt¿/api/posts/get?¡/tt¿ with arguments including ¡tt¿tags¡/tt¿. The return value is also an XML document containing bookmark information. (Note that Delicious offers a number of other API approaches, including JSON. All of these approaches are conceptually similar.) Flickr and Delicious are far from alone in this realm, and we do not mean to ignore the numerous other services which publish human-tagged content: Jots, Linkroll, 43Things, SiteTagger, Lookmarks, Shadows, My-Web 2.0, Digg, Delirious, Creative Mobs are just some of the social tagging web sites that provide tagigng facilities and some form of API.

As a result of these clean data-access APIs, developers have been able to integrate these individual sites into other applications. The Flock web browser comes built-in with Delicious tagging. The Delicious Director project, as well as the more recent Delancey project, offer alternative, more dynamic user interfaces for Delicious bookmarks. Projects like Slickr integrate Flickr images into other datasets, in this case Google Maps. As expected, these open APIs enable the creation of new, useful applications that tightly integrate with this machine-readable data. They decouple the back-end data provider from the front-end user-level functionality.

Even more interesting, given that tagging is a now somewhat-standardized approach to human content categorization, some web sites offer cross-site tag searching. Technorati tag search spans Flickr and Buzznet photos, as well as Furl and Delicious bookmarks. Alternatively, gada.be is a more comprehensive tag search engine which provides results across numerous other tag-managed content sites. In both cases, the new functionality differs from back-end-specific innovations like Delicious Director or Slickr: they combine large numbers of back-end data providers for the ¡b¿same kind of data¡/b¿. They attempt to achieve metadata interoperability: the same meaning across multiple data sources.

One immediately wonders how these tag search sites function. Do they access the API for each site? Thus, do they need intimate knowledge of each site's API, both in terms of procedure calls and return data formats? Clearly, even though tags present one of the simplest possible data

models imaginable, such per-system-consumed cost of aggregation would hurt interoperability.

Thus, tagging has successfully enabled the definition of similar, simple metadata, shared across sites using site-specific APIs. With such APIs, innovation within single applications is rapid and distributed. On the other hand, innovation across many different applications remains an open-ended question: will meta tag searching scale beyond a handful of back-end data sources? Will results be presentable in more aggregate ways than simply listing each data source's results one after the other? Will aggregation be possible on more than the "tag" dimension?

5.2 RSS

RSS is an XML-based data format for linear, timestamped lists of data elements. Most often, these are blogs or other news feeds. Almost every blog software application supports RSS feeds, and new applications have expanded the use of RSS to include the delivery of multimedia files through podcasts and vodcasts. A number of web applications also use RSS feeds to notify users of updates to a collaborative space: Wikipedia publishes its recent changes as RSS, as do collaboration tools like Basecamp.

The history of RSS is confusing and somewhat debated. What's interesting is that most of the debate has centered on the evolution of the schema over time, what specific format it should take, and, importantly, what attributes it might include and how "simple" it should remain. RSS 2.0's approach to schema extensibility is to permit any additions, as long as they are properly scoped using a different namespace. Microsoft has recently released Simple Sharing Extensions (SSE), a set of additional XML elements that extend RSS to enable simple data synchronization.

[[something more about how RSS is effectively a fixed format, even though it sort of allows extensions]]

RSS is thus a very successful data format for exchanging simple metadata. One notes, however, that RSS is fairly fixed in what it can express. One talks of applications that are "RSS-compatible," where RSS is not just the XML syntax, but also the specific schema. Clearly, RSS is limited – by design – in the scope of what it can express.

5.3 RSS and Folksonomies

There is a recent trend to use some combination of REST and RSS to transfer data as categorized by folksonomy tags. In fact, gata.be, the meta-tag search engine, works exactly this way: it knows where to find tag-specific RSS feeds at the various sites it searches, makes the query, and parses the resulting RSS feed.

This technique highlights both the power of RSS and folksonomies as well as their limitations. By bootstrapping off well-established standards like RSS and HTTP, the gata.be designer reduces to a single URL template the incremental work required to add a new source of data. At the same time, this approach is not scalable to more intricate searching, given that the search term is expected to be part of the URL API. In addition, overhead reduction to a single URL is admirable, but it remains unscalable to the wider web given that it requires human intervention.