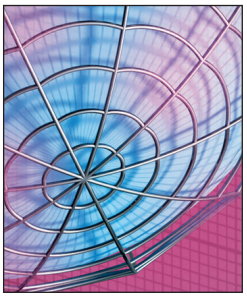
From Here to There



Danny Ayers • *Independent Consultant*

All models are wrong. Some are useful.

— *George E.P. Box*

In the last Websense column, I described where I thought the Web will be headed over the next few years. I speculated that the trend seems to be toward the Semantic Web, although maybe not via the shortest path of directly deploying Semantic Web technologies such as RDF and OWL. For this column, I promised some concrete examples of technologies that support this prognosis.

One key idea of the Semantic Web is the Web of data, in which richly interconnected data collections appear alongside (and integrated with) the current collections of hypertext documents. However, the Web supports linking, and with the various data languages available (often XML based), a Web of data without Semantic Web technologies is entirely conceivable. In a sense, we already have such a thing, although the data are usually binary files such as images and audio files, which seriously limits linking potential.

Without the ability to join pieces of information and work more on the level of knowledge representation, this naïve Web of data offers little promise in itself. There is a possible shift under way, however, from the current Web as (mostly) a document repository with generally limited granularity of addressability, to the Web as a generic, moderately interlinked data store (which includes documents as a subset of data types).

What of Services?

Web (2.0) applications are usually considered services — they perform various functions on behalf of end users. Additionally, we have machine–machine services, such as grid computing systems and those enabled by the Web services specifications (WS-*). Much of the current emphasis on the latter surrounds service-oriented architecture (SOA), which (like Web 2.0) tends to be about reassessing old ideas under a new jargon umbrella. One core idea common to these trends is the development of loosely coupled and interoperable services, which application developers or users can combine into more complex systems. Web 2.0 talks of mashups, WS-* talks of choreography, but the general ideas are the same — services are encapsulated as components, but we can interconnect them through exposed contracts that abstract internal functionality. This is great in principal, and the overall distributed architecture is in line with the Web's. But the devil is in the details.

Some have criticized the WS-* stack because it fails to exploit Web architecture, using HTTP as nothing more than a transport. Web systems interface through a handful of simple methods down alongside the HTTP transport layer. WS-* interfaces are contained in the payload messages — in extreme cases as remote procedure calls (RPCs). This results in increased complexity, but we can offset this to some extent by investing in tools. Web 2.0 systems generally tend to be more faithful to the Web's design, which is an example of the Representation State Transfer (REST) architecture.

However, a fundamental problem — common to most current system implementations following either architectural approach — seriously undermines services' potential benefit. Essentially, the communication between systems is generally based on domain-specific languages. So, although the SOA movement and comparable Web 2.0 APIs have made it easier to arbitrarily wire systems together, once connected, there's no guarantee they'll speak the same language. To use a programming language analogy, the interfaces are strongly typed. This isn't an issue between applications in the same domain, but in a heterogenous environment like the Web, it rules out the possibility of communication between appli-

1089-7801/07/$25.00 © 2007 IEEE

cations that might have some features in common.

The only way forward that enables true loose coupling is to expose the information in a form that allows partial understanding of messages, so that when communication is possible, a potential for crossover across applications exists. This, of course, is a key benefit of Semantic Web technologies: system integration is enabled through using shared, non-domain-specific languages (RDF and OWL). However — and this is important — although the original publisher might expose data in a domain-specific language, there's nothing to stop another party from interpreting that information within a more general-purpose model.

The Transitional Web

The local, immediate lure of machine-processable data's utility has prompted the emergence of technologies that lie between the Web of documents and the generalized Web of data. Microformats (http://microformats.org), for example, allow the publisher to express first-class data in (X)HTML. This data tends to be domain-specific: business cards (hCard), calendars and events (hCalendar), and so on. Microformats make no attempt to generate a general-purpose solution to Web data outside the conventions of specific vocabularies; how systems might interpret individual terms in different contexts or how those terms relate to each other is largely undefined (beyond HTML's syntax constraints).

Looking around the Web (2.0) today, it's easy to see signs of what we could call Semantic Web envy. It's there in systems touted as Web-friendly data stores such as Amazon Simple Storage Service (S3) and Google Base (www.google.com/base/). Bookmark-tagging and ranking sites such as del.icio.us (http://del.icio.us) and Digg (www.digg.com) get their value from (human-generated) descriptions of Web resources. The numerous social-networking systems capture relations between people. Flickr (http://flickr.com) not only hosts photographs, but utilizes personal connections along with tagging. Different data types are integrated locally, but they reach out to the rest of the Web in a very limited fashion, through HTML pages, feeds, and (usually) domain-specific APIs.

Although we might view the "intermediacy" of languages like microformats and Web 2.0 services as a negative, these systems have a winning card because they've actually been implemented. What's more, we can view them as part of the Semantic Web because we can map virtually every kind of data, at least partially, onto the RDF model. Additionally, many of these systems build on existing foundations, rather than reinventing Web infrastructure.

Microformats add extra value to current activities around HTML authoring by relatively transparently including machine-readable data.

Microformats, for example, add extra value to current activities around HTML authoring by relatively transparently including machine-readable data. From a Semantic Web viewpoint, if we can look at it as RDF data, then the particular serialization is irrelevant. For microformats and many other XML-based formats, GRDDL (Gleaning Resource Descriptions from Dialects of Languages; www.w3.org/TR/grddl/) offers a deterministic, unambiguous mechanism for interpreting the material as RDF.

A Web built with explicit data embedded in documents would be considerably more useful than documents alone. It could consistently support the direct manipulation of information as data, without the limitations of "scrape and guess" statistical indexing techniques with human-readable text or purely localized data semantics. In the new scenario, we can view the document in a browser as a user interface to underlying data, which might be human-readable.

Managing Content

With the growth in content management systems (CMSs), control over *microcontent* — subdocument-sized chunks — has increased. *Content syndication technologies* are closely associated with blogs and other news-like information sources. Rather than access to Web pages being relatively passive, with users initiating every request, syndication systems deliver material in a way that emulates the server pushing it. This approach offers significant benefits for often-updated content, in which the end user can subscribe to feeds and receive news updates indefinitely without having to manually navigate to the site in question.

But there's more to syndication than the delivery mechanism. The content being delivered is associated with descriptive metadata, providing information like the an entry's publish date, title, and author. Although it's possible to do this in HTML (using `<meta>` tags, for example), usage varies considerably in "the wild." With syndication, reasonably rich and consistent metadata is baked into the feed formats.

Syndication: An Intermediate Technology

One compelling argument for using RDF is that it lets you say anything

about anything in a common language. Philosophers and logicians might dispute this assertion's accuracy, but RDF is clearly general-purpose, whatever its capabilities. From a content-oriented developer's perspective, however, such generality's benefits are overshadowed by the need for a document-oriented language. Where other explicit data is required, RDF is perceived (often correctly) as having increased complexity compared to a domain-specific format or language. For a typical publisher's purposes, it isn't necessary to say anything about anything, but merely to say a few things about a particular kind of data.

For instance, the RSS syndication language was originally based on RDF, but the typical application of syndication involves reading content through a different interface (the aggregator or newsreader). We can fulfill this application's requirements using a simpler delivery syntax, such as the fork that led to RSS 2.0. This is currently the dominant syndication format. Other factors no doubt contributed to a large number of Web users adopting RSS, but a significant one is that it's the minimum necessary for getting the job done. The "job" here is no longer being part of the Semantic Web, but merely being compatible with existing (and anticipated) newsreading tools. As it happens, due to various failings and ambiguities in the specification, RSS 2.0 turned out to be a little less than what was required to get the job done, but the more recent Atom Syndication Format provides the necessary bug fixes without too much overhead (www.ietf.org/rfc/rfc4287.txt).

Thus, the original RSS was mostly concerned with (meta) data about a site, and the "simple" RSS branch shifted the emphasis to content delivery. This would suggest, if anything, a move away from data. However, syndication's delivery mechanism (polled HTTP) is independent of what's being delivered, and because Atom cleans up the content description, data has returned through the back door. There's also a shift afoot in Atom's other specification, the Atom Publishing Protocol (APP; http://bitworking.org/projects/atom/). This is essentially a protocol for passing content from client to server, typically for authoring applications such as blogs. APP works directly over HTTP, with the Atom format as the primary payload. In addition to create, update, and delete facilities for individual resources, it also includes support for description and discovery of resource collections, and description and discovery of services for manipulating those collections. To support all that, an implementation needs substantial data.

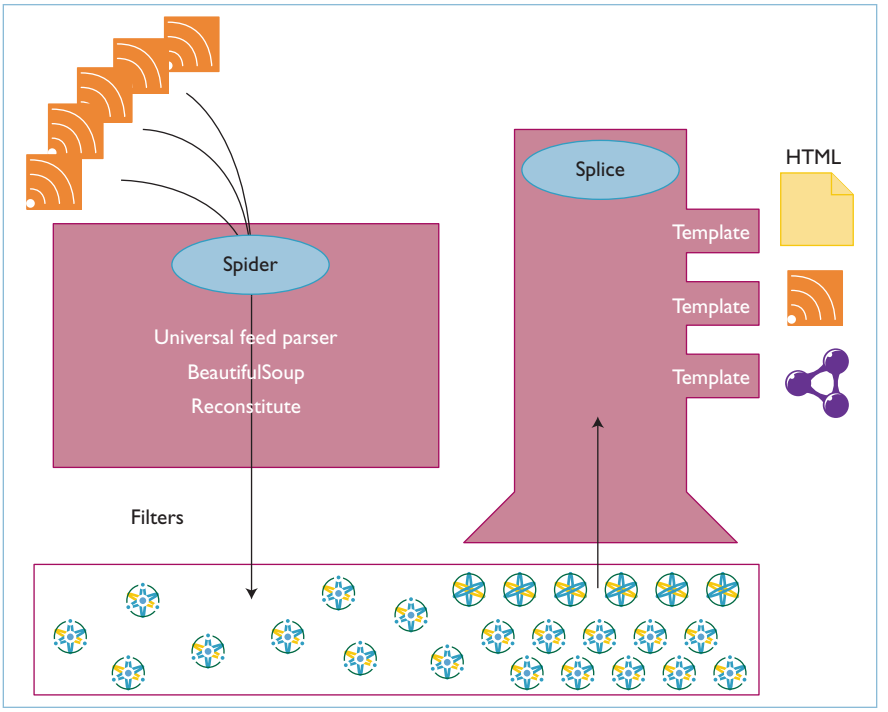


Figure 1. Venus architecture. The system aggregates and filters feed data from the Web, storing it as Atom. The stored Atom can then be remixed and republished in different ways according to templates, such as HTML, feed, and RDF representations.

Just through its existence, Atom implies a certain application architecture. System components have a uniform interface based on HTTP; they're loosely coupled and communicate via APP. These components include newsreaders, which follow the protocol by doing simple HTTP GETs on feed URIs. A service that fully supports the Atom protocol will need some kind of resource persistence with URI-derived addressability; we can generally describe this service as an *Atom store*. Such a store will have a data model based on constructs from the Atom format (feeds, entries, and feed and entry metadata) and APP (workspaces, collections, categories, and so on). Given the protocol endpoints, the Atom system is also well positioned for acting as a data aggregation and processing service. One example is the Venus application architecture that Figure 1 illustrates.

Atom might have the backing of IETF specifications, but a similar service setup is evident in many CMSs. An internal model will have notions of content items and descriptions of those items, along with constructs to enable grouping sets of items. Because a CMS needs to interface with the Web, there's likely to be close correspondence between its internal data identifiers and URIs.

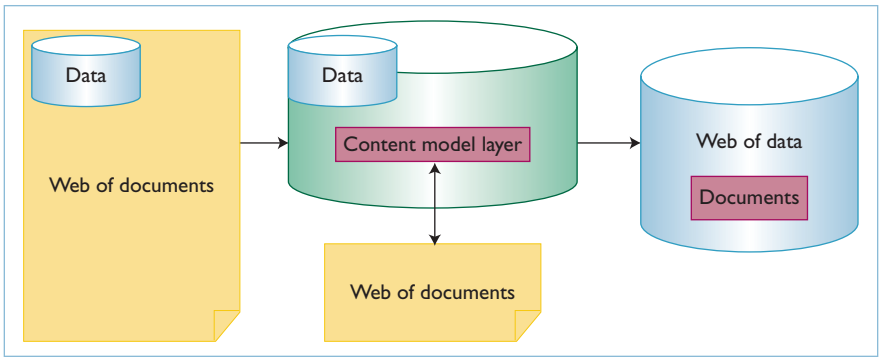


Figure 2. From a Web of documents to a Web of data. Historically, the Web has largely been comprised of human-readable documents, with little explicit data exposed. A smooth transition to a more general Web of data is possible by projecting documents (and their metadata) as explicit data through a uniform data model designed for content.

From the content-oriented Web developer's viewpoint, this approach is nothing special — again, it's just getting the job done. For those who care, however, using a common model makes building interoperable services easier. Similarly, for the data-oriented Semantic Web developer, at least at first glance, there isn't much to get excited about — this approach deals with human-readable content in a non-RDF model with non-RDF syntaxes. But although these systems might not use RDF, they're built squarely around resource description. The language used isn't generalized to enable description of anything — just content and content collections. But within that context, it's as general-purpose as you can get, while still being optimized for the specific (content) domain. Atom is a domain-specific XML dialect, but automated systems can map it to the RDF world.

A Layer Between

An aspect of many recent content-oriented technologies is worth particular attention: for want of a better name, I'm calling it the *content model layer*. The general idea is a data model oriented toward content. Such a model is described in the Atom format/protocol specification, the Java Content Repository (http://jcp.org/en/jsr/detail?id=170), and numerous other systems on the Web today. The content model layer treats documents and their associated metadata as a unified whole. Obviously, this isn't a new idea (given that the systems I mentioned are already implemented), but treating this model as a layer in a stack of data representations provides a different perspective on the idea's potential.

Layering is well known as an effective route to complex system development. Each layer is dependent on the layer below it, but to layers above, it provides a simplified abstraction of everything below. Ideally, the abstraction layers won't "leak" (reveal what's under the hood), and the list-like structure will provide some separation of concerns. Content and its metadata on the Web have various parts; some are identified with URIs, and some involve special names such as the content media type (text/html, image/jpeg, and so on) and various other HTTP header fields. The additional metadata is usually based on a fairly narrow set of concepts, such as the publication date and authors.

Again, this isn't new — people have been working with documents in this general fashion for as long as information has been recorded. The reason I think the approach is particularly useful at this point, however, is that we now have the conceptual (and programming) tools for the Semantic Web — a generic model of information — and the documents in question are already on the Web. The real-world librarian might use a book author's name or its Dewey categorization to determine on which shelf to place it. But shifting up a layer of abstraction, how do you usefully shelve documents in a completely generalized, massive-scale, interconnected data repository? We're no longer looking up from the document to the document repository, but from the knowledge representation model downward. The content model layer offers a view of the content (and its metadata) at a level of abstraction that works from the current document-based Web's perspective, from the perspective of a naïve Web of data, and from the perspective of Semantic Web languages. Figure 2 shows how the content model layer projects documents onto a Web of data.

Incidentally, it's been suggested that RSS itself could become the lingua franca of Web data. Although adding RSS's (comparatively) standardized metadata to messages of any kind would allow a degree of uniformity in data handling, taken alone, this wouldn't be a great advance on the naïve Web of data we have now. Instead, it would be integration at the lowest common denominator rather than the highest common factor. But in a larger context, within a fledgling Semantic Web, Web applications could integrate this kind of material with other data types from other sources, and the potential benefit would be amplified by the network effect across domains.

Data all the Way Down

Even without the Semantic Web as the specific target, if you extrapolate from recent developments, some kind of Web of data seems fairly inevitable. Different paths are evident: exposing existing off-Web data, adding explicit data to content, or treating content as data. The latter has had little analysis, but I believe it's reasonable to consider treating content as data via initiatives like Atom as an emergent

increment between the Web of documents and a Web of data. Within the context of content-oriented systems, integration comparable to the kind that Semantic Web technologies offer is possible using a model that's general-purpose within that scope.

To the Semantic Web enthusiast, this might sound painfully suboptimal. But the fact is that developers care most about solving local problems in the easiest possible way with tools they know. That rarely means looking to global solutions because they usually involve extra work. Semantic Web technologies are still perceived as being complex, and their biggest benefits accrue from being part of a network including third-party tools that use those technologies. Perceptions take time to change, and adoption doesn't happen overnight.

Developers' experiences suggest that the demands of the Semantic Web vision are generally well-aligned with Web systems' practical requirements. As mentioned in reference to SOA and Web 2.0, it's generally getting easier to implement intercomputer connectivity and distributed computing architectures using the Web as a platform. Yet the ability to connect and distribute data is of limited use without some way to intelligently integrate that data. Fortunately, Semantic Web technologies can increase the potential for data integration. Maybe future events will lead to the rapid, viral spread of these technologies, but that seems unlikely right now. Immediate benefits come from leveraging what we already have, and that means primarily a Web of documents. Still, it seems likely that with developers maximizing the benefits through data-oriented Web systems, momentum is likely to carry us to a Semantic Web sooner rather than later.

Danny Ayers is an independent developer, consultant, and author. His research interests are primarily around Semantic Web technologies. Ayers has coauthored 10 books on programming, generally covering Web-related topics. He is chair of the Developers Track for WWW 2007. His Weblog is at http://dannyayers.com. Contact him at danny.ayers.ieee@gmail.com.