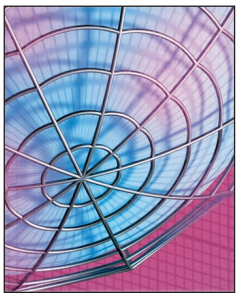

Graph Farming

Danny Ayers • *Talis*

In my last column ("A Difficult Abstraction," vol. 11, no. 5, pp. 86–89), I attempted to demonstrate that the level of abstraction that most Web developers work at isn't well suited for increasing the Web's sophistication and hence its utility to end users. The specifications are such that either things get hard very quickly, or developers take shortcuts, employing ad hoc abstractions that diverge from Web standards. In this issue, I'm going to talk more about the Web's evolution (in an increasingly roundabout fashion), and then offer some suggestions toward another abstraction of the Web that I think might alleviate many current obstacles to a significantly better Web.

The Giant Global Graph

Since my last column, Tim Berners-Lee has obligingly published three items I believe support the arguments I made to some extent. In a new Design Issues post, "Levels of Abstraction: Net, Web, Graph" (www.w3.org/DesignIssues/Abstractions.html), he describes conceptual abstractions — from the wires of early telecommunications, through computers, to networks and the Internet, and on to the Web. The level of abstraction he emphasizes is a change in focus from a Web that's about documents to a "graph" that's about things (objects, people, places, concepts, and so on) and the connections between those things. This is the level at which Semantic Web technologies should let us operate.

This graph-of-things abstraction has featured in Berners-Lee's work for a long time (a good example is his presentation from the first International WWW Conference in 1994 at www.w3.org/Talks/WWW94Tim), but his motivation for reiterating the idea now is clear in a blog post on the subject entitled "Giant Global Graph" (http://dig.csail.mit.edu/breadcrumbs/node/215). In it, he explains how the phenomenal growth of social networking sites has recently prompted deeper discussion of the real-world social networks they describe. These sites are in effect closed worlds, with little cross-system interoperability. In a piece which created quite a stir in the blogosphere, "Thoughts on the Social Graph" (http://bradfitz.com/social-graph-problem), Brad Fitzpatrick and David Recordon identify the problems encountered when interacting with systems built on silos of personal data, the disparity between these systems, and potential solutions. They've brought the *social graph*, the "global mapping of everybody and how they're related," to wider consciousness. But as Berners-Lee points out, this interpersonal graph is only a subset of a greater graph of things.

I believe compelling evidence exists that the global graph abstraction Berners-Lee describes is key for the Web to thrive long term, although his word choice is slightly daring. The phrase "Semantic Web" is a reasonably accurate reflection of what it describes, and isn't likely to go away, yet probably wouldn't have been a marketer's first choice. Some in the Semantic Web community have substituted "Web of data" wherever possible in discussions in which the original phrase might have been misconstrued. Personally, I like this phrase because it nicely contrasts with "Web of documents," and if you make your data Web-shaped, you have the very same global graph as Berners-Lee discusses. Others are willing to grit their teeth and talk of "Web 3.0" but aside from aesthetic considerations, this brings in difficult questions of scope. However, although a Giant Global Graph might work as a valid, useful, and palatable high-level view, it's far from the end of the story for people involved in building this stuff.

In my last column, I included a few diagrams I hoped would show that protocols such as HTTP

 1089-7801/08/$25.00

weren't exactly trivial even in a document-oriented context, and that when the graph model of RDF (the foundation of Semantic Web technologies) was brought in, it pushed the limits of what any mortal developer could be expected to comprehend, let alone use in his or her day job. The final new item from Berners-Lee that I'd like you to note is an image he's created that blows my sketches out of the water. His "The basic follow-your-nose way the Web works" diagram (www.w3.org/DesignIssues/diagrams/arch/follow.png) shows the core components involved when traversing a Web of graph-shaped data and documents. He's elided subsystems here that would significantly increase the diagram's complexity — for example, the potentially recursive Gleaning Resource Descriptions from Dialects of Languages (GRDDL; www.w3.org/TR/grddl-primer/) mechanism for interpreting XML documents as RDF. Yet it's already intricate enough to suggest it might make a good diagnostic for Asperger's syndrome (and membership to certain W3C Working Groups). I'd like to suggest an abstraction that I think might help avoid this kind of diagram's cognitive overload, but let me first provide some context.

Bits and Beans

A common phrase in Web specifications is *user agent*, defined in HTTP 1.1 (www.w3.org/Protocols/rfc2616/rfc2616.html) as "The client which initiates a request. These are often browsers, editors, spiders (Web-traversing robots), or other end-user tools." To put this into context, consider a typical set of Web interactions. Say you're getting a little hungry and need a nutritious bean recipe. You open a browser and visit a search engine to get a list of pages likely to contain the kind of dish you have in mind. You visit the most promising page, then make haste to the kitchen. Your browser has acted as a user agent here, intermediating with remote services on your behalf.

Now consider the busy developer's solution, a can of baked beans. Simple, nutritious, convenient food. Open the can, gently heat the contents with your preferred cooking facility, serve on toast — delicious. As the product's end user, your interactions with it prior to consumption are fairly limited — probably involving the choice of product, a financial transaction in a supermarket, and transport to the pantry. In terms of information complexity, introducing the beans to toast seems comparable to finding a recipe on the Web. The beans were once a remote resource; the supermarket acted as a real-world user agent to provide them to you.

But what of the system complexity beyond the user agent? The recipe page's creator likely used a content-management system to present the text in an appetizing form. The recipe document sat in a database or filesystem behind a Web server until the appropriate protocol requests came along, at which point, the server delivered the document to the user agent. The search engine that enabled identification of the resource of interest did a lot of work behind the scenes: it spidered and indexed a large set of documents in advance, so that when the query came in, various algorithms selected, ranked, and returned pointers to documents that corresponded to the query.

This is, of course, a hugely simplified view of the system server-side. Several protocol layers will have been in action, notably TCP/IP, DNS, and HTTP. The packets conveying the information from the remote server to your user agent might have come a long and convoluted route through many intermediaries. To be useful at Web scale, a search engine is going to be nontrivial. Chances are the search engine will use some kind of distributed architecture to balance the data and computational load across multiple physical systems, with several layers of abstraction used to make the system work from the human query-response view right down to the flipping of individual bits. This is a software system; for the sake of argument, let's not go into the hardware.

So what's server-side for the can of beans? The most local level of transport will have been relatively straightforward in terms of conceptual complexity, probably shipping by road and rail. But the biggest difference here is in how the can of beans came into being. There are the vegetable materials that wound up on the end user's plate: beans, and whatever went into the sauce. It seems reasonable to consider the bean plants as the origin level in this scenario. But the packaging is part of the product, too. There are numerous long paths back to the constituent components. Not only were the beans and tomatoes nurtured on the farm before harvesting, the can's metal was mined and the iron smelted into steel before pressing and plating. Even the label's creation will have been nontrivial, with wood pulp probably bleached by chlorides before pressed into paper, decorated, and protected with colored varnishes of petrochemical origin.

Although the general amount of end-user interaction is similar in these two scenarios, if you drill down and open the hood on some of the abstractions, much more complex and diverse components are at work behind the material production and delivery system than the information production and delivery system.

Putting aside any social and political questions, the consumer-oriented material systems we have in this day and age are in reality composed of several different components, interconnected through social and economic protocols, along with plain old physics. The scale and complexity of individual components var-

ies, but we can demarcate each as a commercial unit. We can view these units as acting as discrete agents, communicating through common, standard interfaces (typical physical transport and money).

The point I'm trying to circle around here is that compared to typical real-world systems, the systems currently operating on the Internet are remarkably crude. We do have the low-level software and hardware infrastructure available to support anything we like at a global scale. Take the One Laptop Per Child initiative (http://laptop.org) — it's at least broadly conceivable that we could provide every child on the planet with a computer that's connect to the Internet. But as an information society, our daily individual interactions with the network are still at the hunter-gatherer stage. This is only to be expected given the Internet's novelty. Fifty years ago, it might have been broadly conceivable that we could provide every child with an individual book. Now we're talking about access to every single book written, plus a whole lot more, with Berners-Lee's Giant Global Graph — and keep in mind that computers can do more than act as dumb conveyers of data; they can compute. What we need here is a way of thinking about Internet-based systems that supports the Giant Global Graph abstraction, encompasses processing in this environment yet is conceptually simple enough that real-world developers can use it in their daily work.

Bring on the Agents!

Not long ago, James Hendler asked, "Where Are All the Intelligent Agents?" (*IEEE Intelligent Systems*, vol. 11, no. 3, 2007, pp. 2–3). The article's abstract is succinct: "There has been much research and talk about intelligent agents, but few real-world implementations." I'd like to attempt to answer Hendler's question in the context of the Web.

One possible answer is that, in the new environment, the agents simply got renamed "services." A Web service typically takes input from the outside world, carries out operations locally, and returns the desired output. It acts as a quasi-autonomous system, and can interact with other such systems in the same general manner as traditional agents. This would be well and good but for developers' tendency to use simple abstractions that skip features of the Web specifications or develop systems that explode in complexity with little hope of rich interoperability with other Web systems. Either way, almost inevitably the immediate motivation of providing some local functionality within the usual constraints overwhelm considerations of the bigger picture. I say almost inevitably because there is a proportion of system architects and Web developers that realize the value that high levels of interoperability with the outside world can offer, even if that value is hard to measure in advance.

The Giant Global Graph is an excellent perspective on how we can consider diverse pieces of Web infrastructure as a conceptual whole. The Web, when augmented with RDF's graph model, provides an interconnected system. The resources on the Semantic Web are interconnected through logical predicates, but this layer exists on top of the coupling offered by the link, as built into the Web since day one. Resources can be identified with URIs designed for HTTP, and the primary Web mechanism is that of using HTTP to get representations of those resources. Put this together and you have a system in which you can follow your nose through links of interest to find more related information from any given point. This approach is exactly the same in principle as the document Web's linkage and navigation. However, when the material being traversed contains a reasonable proportion of machine-readable semantic information (rather than text and media that only a human can decipher), the potential for software agents to act autonomously is greatly increased. It was while working in this context that I found myself with code that hinted at a simple abstraction for Web systems.

What the Code Suggested

A couple of years ago, largely out of curiosity, I worked on some RDF-based syndication code. I had lots of directories full of Python scripts, and it was getting unwieldy. Around the same time, an employer asked me to investigate the Java Agent Development Framework (JADE; http://jade.tilab.com). I found it an eye-opener. Here was (an implementation of) a conceptually simple abstraction of software systems that felt very natural. For any given application, you have a number of agents acting autonomously, each with its own set of localized behaviors and the ability to communicate with other agents. I was quite surprised to realize that unbeknown to me, my Python code had been heading toward this same abstraction. My scripts were relatively autonomous; they each had a small set of behaviors and an application would be constructed by wiring the individual agents together.

Although my "agents" did most of their intercommunication through an RDF store acting as a whiteboard, they communicated with other systems over HTTP. Figure 1 shows what I believe is a feasible general model for a class of agents on the Semantic Web.

The idea is that like most traditional artificial intelligence agents, these hypothetical Semantic Web agents interact through message passing, in this case with the Web as the medium for those messages. They have local memory (the RDF model), the size of which will depend

on what's required to support their behaviors. Although for many data routing and processing tasks, only a small amount of immediately accessible data will be needed, agents that are intended to provide long-term memory — for example, to act as the backing store for a traditional content-oriented Web site — will allocate whatever space they need. Within the agents' memory will be self descriptions and more than likely a cache containing information about agents with which they commonly interact. The behavior will ideally be limited to some discrete piece of functionality, aggregating or filtering data from other sources, for example. The HTTP server subcomponent exposes all the resources in the agent's immediate scope to the Web, allowing the internal state to be transferred. The HTTP client allows the agent to get more information from other agents on the Web. A common feature among such agents would be behavior to provide (and interpret) data in representations that aren't RDF-oriented. Such behavior could be implemented locally, or it might be preferable to allocate agents to specific representation-conversion tasks. In practice, it's likely that systems built on this kind of agent would use auxiliary storage for "pure" content — static representations that rarely change. Alternately, a direct connection could be made between the agent's internal RDF model and external non-RDF material for simple representation routing or for on-the-fly data translation.

However, a potential conflict exists between an architecture like this and typical Web components. HTTP is request–response; a prerequisite for useful messaging is the ability to act asynchronously. Agents might be required to communicate with other agents for their work, yet still respond in reasonable time to any request. Here, I defer to the "Behavior" black box in the figure — where necessary, another process would be spawned internally and use the HTTP client facility as required for other interactions.

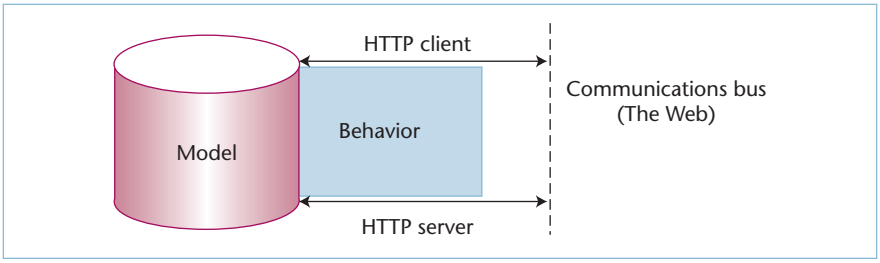

Figure 1. Semantic Web agents. An agent has a working memory (in the RDF Model), input and output through HTTP connectors, and behavior which defines how it processes information.

Architecture, Implementation, or Abstraction?

Okay, so I cheated — the kind of agent system I've described could also be considered an architecture for developing applications from scratch. In fact, I coded up a fair amount of such a system (and intend to work on it again when time permits; it's an irresistible exercise), quite enough to identify some of the immediate issues — for example, it calls for serious contortionism to make HTTP client and server components seem like messaging inputs and outputs. The not-quite similarity between object-oriented classes and instances (and running processes) of the programming language and those of Semantic Web languages caused a fair amount of confusion.

I thought it necessary to present this architecture as an agent design in part because the notion of "user agent" we've carried over from the Web's early days is somewhat back to front compared with what's needed for a more sophisticated Web. Rather than conceptualizing clients as agents for end users, it seems to allow far greater versatility if we think of them as agents that act like autonomous users, interacting with other agents on the Web. I'm not suggesting full-blown, belief-based, intentional agents. Rather, I'm suggesting that we refactor our view of the systems we're already using to see them in a more self-contained, discrete fashion. Exposing a database as linked data or enriching a content management system with Semantic Web capabilities could be seen as a special case of the RDF-oriented agent.

The document Web's preoccupation with the browser is entirely understandable given its role as a document viewer, but perhaps other areas need more attention. The client-server abstraction itself leans us toward a two-ended vision of how the system as a whole operates — there's the service, here's the end user (or worse still, vice versa). The service might have multiple tiers behind it, but that doesn't help much where it really matters on the Web, close to the links that connect it to the rest of the graph.

In the real world, I have no knowledge of the system that grew the beans that went into the can, nor of the steel manufacturing process that went into making the can. The bean growers and steelworkers have little need to interact either. Yet, user interactions with the Web are predominantly like hand-to-mouth existence. We need to move on to information agriculture.

Danny Ayers works for Talis as a developer community liaison for its Semantic Web platform (http://talis.com/platform). His blog is at dannyayers.com. Contact him at danny.ayers.ieee@gmail.com.