9-2007

The Programmable Web: Agile, Social, and Grassroots Computing

E. Michael Maximilien

Ajith Harshana Ranabahu
Wright State University - Main Campus

The Programmable Web: Agile, Social, and Grassroots Computing

E. Michael Maximilien
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
maxim@us.ibm.com

Ajith Ranabahu
Wright State University
3640 Colonel Glenn Highway
Dayton, OH 45435, USA
ajith.ranabahu@gmail.com

Abstract

Web services, the Semantic Web, and Web 2.0 are three somewhat separate movements trying to make the Web a programmable substrate. While each has achieved some level of success on its own right, it is becoming apparent that the grassroots approach of the Web 2.0 is gaining greater success than the other two. In this paper we analyze the movements, briefly describing their main traits, and outlining their primary assumptions. We then frame the common problem of achieving a programmable Web within the context of distributed computing and software engineering and attempt to show why Web 2.0 is closest to give a pragmatic solution to the problem and will therefore likely continue to have the most success while the other two only have cursory contributions.

1. Introduction

There is wide agreement and efforts toward expanding the Web into a platform and substrate for distributed computing. To that end, three separate movements and approaches have started around the turn of the century.

First, the *Web services* [1] movement, which is backed by various members of industry and some from academia. The effort in this area resulted in a plethora of specifications, commonly called WS-*, that attempt to create a complete distributed programming enterprise stack around XML and the Web. Various industry leaders have invested heavily in the creation of this architecture and to provide tooling, training, and solutions centered around Service-Oriented Architectures (SOAs) and Web services [3]. While this effort is the most mature and complete, it is arguably the most complicated. This perceived complexity has slowed down its widespread adoption and deployment.

The second movement, or *Semantic Web* [2], has its root in academia and in particular the field of artificial intelligence. Widely advocated by the W3C and various prominent scholars, the Semantic Web is described to simply add meaning to the numerous heterogeneous data, services, and content on the Web via semantic annotations (or metadata). Since the metadata is created using variants of description logics, one can create software agents that can use the additional *data about data* to infer various interesting properties of the data and services as well as help facilitate the usage of the annotated content and services.

Finally, the third movement or *Web 2.0* [14], makes use of Web's ability to be a social substrate and extending it to be a social programming substrate. This movement lead to the creation of various socially-oriented applications such as MySpace.com, Digg.com, and TechCrunch.com. These sites share one thing in common, and that is, they help connect individuals and also facilitate their collaborations. Along with this emerging *social Web* the facilities afforded by new programming paradigms (e.g., JavaScript with AJAX, Ruby with Ruby on Rails, and PHP, and various new simpler XML applications for consuming data and services) are allowing these Web 2.0 applications to be exposed as APIs for remixing or for the creation of novel composite applications, i.e., *mashups*.

While the three movements have one thing in common: the Web as a programmable platform, they each took radically different approaches to how they achieve and evolve this platform. It maybe too early to declare a clear winner, however, the momentum seems to be with the Web 2.0 movement. In this paper we attempt to provide a critique of the Web services and Semantic Web efforts and provide a rational on why current dynamic and agile Web 2.0 programming approach has less need for the Web services standards and for semantics than might appear on the surface.

1.1. Organization

The rest of this paper is organized as follows. Section 2 briefly presents SOA and the Web services architecture

[1] http://www.w3.org/TR/ws-arch

highlighting some of its main flaws and a rational as to why it has become so complex. Section 3 summarizes some of the promises of the Semantic Web and also raises some of the problematic assumptions that it puts forward. In Section 4 we give a abbreviated discussion of the problem at hand: namely transforming the Web as a programming substrate. We then draw a potential solution in Section 5 based on current Web 2.0 trends but also using evidences from eminent software engineering principles. Finally, Section 6 finishes with a short conclusion.

2. Web Services and SOA

The first widespread effort to transform the Web into a programmable substrate for creating and integrating applications started with the Web services efforts. The basic architecture started as a simple model of distributed computing using XML for data modeling and exchange. Briefly, service providers expose their services described using WSDL [2] that they publish in registries called UDDI [3], which are then found by service clients and consumed using the SOAP [4] protocol.

2.1. WS-*

This basic architecture (without the specific implementations) is seen as the core of SOAs and calls for granular, loosely coupled, and XML-based access to business processes and data over a network. While simple in its pure form, the Web services incarnation of SOAs started to address non-functional issues (e.g., security, scalability, agreements, policies, transactions, and so on) that would help make Web services a complete enterprise computing stack.

The WS-* set of standards was created with the primary objective of having a complete enterprise software stack that would help solve all of the basic issues experienced in enterprise software systems. Various proposals addressing security, reliability, agreements, negotiations, composition, and so on, made it to the lime light in an effort to help standardize the stack and avoid its potential fragmentation.

2.2. Sources of Complexity

While the WS-* stack has fundamentally addressed various aspects to make the Web an enterprise-ready platform, they have also attracted many criticisms [15]. Some of the descending views centered around the perceived level of complexity when trying to implement and use the proposals.

But *what is the source of the complexity* in the WS-* proposals? Surely, the creators of these proposals have many years of experiences building distributed systems and are highly qualified and must have thought about the beauty of simplicity vs. the hell that comes with complexity... We believe the problem is not with the validity of the standards but rather a scoping problem and lack of foresight.

First, the Web services effort tried to address the entire spectrum of features and guarantees that need to be addressed by enterprise IT. The intent is to allow architecture and implementation of enterprise applications (with all or part-of of common enterprise-ready features) using the new Web stack. While a noble effort, this leads to a huge effort with broad scope. The basic SOA architecture required various heavyweight middleware to get even the simplest applications implemented.

Finally, the Web is dynamic, evolves constantly, and is above all an open platform. These fundamental characteristics have significant impact on the success of proposed Web standards. In particular, it's easy to observe that most successful Web standards (other than the basic ones) tend to evolve via a darwinian mechanism similar to the open-source software (OSS) movement [11, 4]. The Web services effort was slow to adopt these governance practices and instead evolved mostly by committee.

3. The Promise of Semantics

The other planned effort to make the Web a programmable platform has its roots in artificial intelligence and primarily the subfields of knowledge representations and description logics. The Semantic Web and many of its proponents advocate that a key problem to the evolution of the Web is the lack of expressed semantics. Since the Web is primarily a collection of heterogeneous data (or content) and service, which are also offered, by a no-less heterogeneous set of agents, there are bound to be inconsistencies and contradictions. The vision of the Semantic Web is that the *meaning* of the content and services can be mostly expressed explicitly using annotations over common and agreed upon shared conceptualizations, i.e., ontologies [7].

3.1. Disconnect and Return on Investment

While the Semantic Web has sound foundations, the lack of widespread adoptions and applications to real-world scenarios suggest that it is somewhat disconnected from the realities of the Web. In particular, one of the Web's strengths is its inherent heterogeneity and dynamism. These traits reflect the heterogeneity and dynamism of society and of the various human cultures in existence. By requiring ontologies to enable knowledge sharing and for annotations, the Semantic Web imposed a heavy burden on designers and on interoperability since consensus building and precise knowledge engineering are hard tasks.

[2] http://www.w3.org/TR/wsdl
[3] http://www.uddi.org
[4] http://www.w3.org/TR/soap

Further, even if ontologies were widely available and used, there is still the pragmatic issue of *return on investment* (ROI) for companies (large or small). Building or reusing an ontology to describe ones content or service has little bearing on how fast or easy the content or service can be implemented or maintained. Additionally, there is little evidence that taking the time to describe ones service or content semantically on the Web facilitates integration or usage. These latter aspects are highly promoted as being one of the key values to the Semantic Web, however, outside of small academic circles and specialized problems and domains, not much evidence have surfaced

3.2. Problematic Assumptions

One way to understand the Semantic Web's lack of success is that it makes a series of problematic assumptions.

First, shared agreed upon conceptualizations are difficult to achieve and the return on investment is very small. The ROI won't get better until others make use of the shared conceptualizations, in other words until *network effects* [5] take place.

Secondly, while semantic annotations of content and services promise much future rewards, they are difficult to achieve centrally (sheer size of the job) and lack incentives for distributed annotation solutions. Indeed, a Web user annotation of Web site's content or service is done primarily to ease searches and for personal categorization and are idiosyncratic. Further, modern search engine's high accuracy and recall have also made this task less relevant. A few Google searches typically yields the Web content or Web service that a user had in mind.

Finally, Semantic Web technologies were designed to provide higher levels of abstractions that disconnected them to the realities of programmers. The disconnect occurs on two levels:

- First, most Semantic Web technologies (e.g., RDF [6], OWL [7], SPARQL [8], and so on) are removed from the programming languages that most Web developers are used to. They require translations which typically leads to round-trip engineering and tweaking. This is akin to model-driven software architecture and development [8], which similarly has had difficulties gaining widespread adoptions.
- Second, because Semantic Web technologies require heavy investments in up-front conceptualizations and modeling they are not *agile* [1] in the sense of achieving quick, malleable, and executable systems that can be shown in an iterative manner to end-users.

[5] http://en.wikipedia.org/wiki/Network effect
[6] http://www.w3.org/RDF
[7] http://www.w3.org/2004/OWL
[8] http://www.w3.org/TR/rdf-sparql-query

4. Back to Reality

In many ways, the Semantic Web and Web services have strayed away from the realities of the Web as we know and use it today. First, in the case of the Semantic Web, the call for up-front conceptualizations and agreements are almost antithesis to the Web's success, governance models, and use. Besides some basic protocols and initial standards, Web programmers and site designers and developers are free to create whatever they like, when they like. This freedom of expression, akin to democratic virtues, seem to bode well with humans of all races, creeds, and background. There is something empowering to being able to contribute content or expose services that can be widely accessed without first requesting permissions or agreeing on common terminology. Indeed businesses are thriving on the Web due to the fact that they can re-implement and re-engineer their assets and businesses quickly, efficiently, without any concerns from competitors or the domain that they belong. They are free to take advantage of the Web as and when they see fit.

In the case of Web services, the resulting standards, though well thought out in some aspects, have failed to address the basic needs of Web programmers who have to deal with increased pressures of quick delivery for mostly unstable requirements. Indeed, the Web has made the need for agile development methodology and frameworks a primary concern among Web programmers [12, 13]. The Web services standards are difficult to directly program in. They require various levels of tooling to translate the artifacts of the specifications to what is manipulated by the underlying programming language, e.g., service descriptions using WSDLs.

4.1. Framing the Problem

To better attempt to understand the problem of using the Web as a programmable platform, let's try to frame the problem by highlighting the main requirements and facilities that such a platform should offer.

1. *Built on Web standards* such that there would be no (or minimal) need to create new Web protocols or standards. In many ways, this may appear (in hindsight) an obvious requirement. The Web as we know it know does a fairly good job at providing a platform for Web applications that are becoming richer and richer (via AJAX [9]) and also more interactive. Why should a programmable layer over it be any different?
2. *Ease of programming*. This is key in our opinion. There is no need to introduce additional complexity to programming Web applications, services, and systems. Any new frameworks or concepts should blend

into the current toolsets that programmers already have and are used to. Naturally, this does not mean avoiding creating new languages or frameworks, but instead it's about evolving or improving what is currently available without requiring the use of new higher-level representations that need levels of translations before executing.

3. *Facilitate collaboration* to accommodate and to take advantage of the nature and affinity of the Web as a platform for human social networking. Programmers and designers also collaborate. The success of the OSS movement is clear evidence that collaboration is key to allowing any significant piece of software, architecture, or concept to move beyond niche acceptance to widespread usage on the open Web.

4. *Language and framework agnostic* is also an important characteristic. While some languages (due to their abstraction capabilities) will have better affinity to Web programming or make that process relatively easier, there is no need to advantage one language over another.

5. *Agility* is paramount. This means having a platform which respects and encourages the accepted practices of agile software development [12]. In particular, there is no need to encourage big design up-front (as in ontologies) since such designs rarely remain valid during the life of the system. Other important agile principles to facilitate are enabling short code-test-deploy cycles and minimize non-executable artifacts.

6. *Solution to common distributed programming issues*, that is, attempt to provide a platform that has primitives to address the common issues encountered in distributed systems, e.g., latency, failures, security, and so on.

4.2. Issues with Distributed Programming

The *fallacies of distributed computing* [6], first identified by Peter Deutsch of Sun Microsystems, are a common set of false assumptions that are typically made about systems that have distributed nodes of computations. Nowadays we typically count eight fallacies:

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology does not change
6. There is one administrator
7. Transport cost is nil
8. The network is homogeneous

The Web already does away with some of these assumptions and as a platform it needs to strive to not to make these assumptions and address the true issues that they pose. Furthermore, the platform should encourage frameworks and languages that help programmers and designers create systems that do not make these assumptions and deal with the issues directly, e.g., asynchronous calls via AJAX to deal with latency.

5. Social, Grassroots, and Agile Solution

The Web 2.0 movement is the prototype of what is necessary to achieve this programmable Web. The success of this movement is not only due to the various new social applications that it enables but also due to the grassroots community that govern its evolution and to the agile software engineering approach that it encourages.

5.1. Software Engineering

In many ways, creating a programmable substrate using the Web amounts to realizing a platform that addresses basic software engineering principles. Semantics can be useful for conceptualizations and modeling, but at the end of the day the problem remains a software engineering problem [10]. The Web 2.0 efforts around using the Web protocols and URLs to expose business functions as REST [5] services and data services as Atom [9] or RSS [10] is the realization of this idea in its simplest form.

Furthermore, the platform must also be agile. In particular it needs to enable rapid creation, test, and deployment of Web applications and services. Example of this agility can be seen in the Web 2.0 with the advent of service mashups that are prototypical examples of rapid compositions and creations of new applications using a combination of different other services.

The agility is also present in how Web modeling are achieved. Unlike heavyweight approaches that tend to require big up-front modeling investments, in this platform, programmers are able to create simple and efficient models for their data and services as they experiment using them on the Web, e.g., JSON [11] and YAML [12] for data representations. There is a very low impedance mismatch between

[9] http://en.wikipedia.org/wiki/Atom (standard)
[10] http://en.wikipedia.org/wiki/RSS
[11] http://www.json.org
[12] http://www.yaml.org

the models and the executable code. In many ways the models of the data and semantics of the services are part of the code that realize the services.

5.2. Social and Grassroots Computing

As shown by the most successful software endeavors on the Web, such as those of the OSS movement, enabling social computing is fundamental to making the Web a programmable platform. Examples of this social and collaborative platform are being realized now with wikis and tagging which allows humans to contribute knowledge and some level of idiosyncratic semantics (also known as *folksonomies*) to Web content, resources, and services, e.g., Wikipedia.org, Flickr.com, and YouTube.com.

Additional efforts could be in providing example usages (use cases) for the available services and content. This allows easier reuse of services in mashups but also help create communities around the services and help foster their evolution.

6. Conclusion

In trying to achieve a programmable Web, it's important to recall basic distributed computing concerns, the roots of the Web, and basic software engineering principles. Like other distributed systems before it, the basic concerns of latency, errors, security, and so on, have to be addressed. However, unlike other systems, the Web's primary focus has been to create the connection between the Internet and end-users. This implies that the Web as a platform should stem from this human connection first. Web 2.0 has benefited from tackling this aspect and by focusing on socially connecting humans.

Additionally, using simple programming paradigms that take full advantage of the Web's architecture and with simple data models, Web 2.0 APIs and mashups are achieving more success then the thorough, heavy, and complete WS-* standards. This success has a lot to do with keeping the service models simple and nimble.

Finally, from a programmable viewpoint the Web 2.0 platform is agile at it's core. It focuses on creating artifacts that are programmable, lightweight, and directly usable. Unlike the Semantic Web, there is very low entry point to using the Web 2.0 artifacts. There is no need for big design up-front and instead, Web 2.0 is grassroots. Data and service semantics are understood and communicated using examples and use cases. This leads to a platform that is dynamic and agile and programmer friendly.

References

[1] K. Beck and C. Andres. *eXtreme Programming Explained: Embrace Change, 2nd Edition*. Addison-Wesley, Boston, MA, 2005.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 501(5):28–37, May 2001.

[3] T. Erl. *Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Indianapolis, IN, 2004.

[4] J. Feller, B. Fitzgerald, and E. S. Raymond. *Understanding Open Source Software Development*. Addison-Wesley, Boston, MA, 2001.

[5] R. T. Fielding. *Software Architectural Styles for Network-based Applications*. Ph.D. thesis, University of California, Irvine, CA, Jan. 2000.

[6] M. K. Goff. *Network Distributed Computing: Fitscapes and Fallacies*. Prentice Hall, Upper Saddle River, NJ, 2003.

[7] T. R. Gruber. The Role of a Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In *Proceedings of the Knowledge Representation and Reasoning Conference*, pages 601–602, 1991.

[8] B. Hailpern and P. Tar. Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, 45(3):451–462, 2006.

[9] M. Mahemoff. *AJAX Design Patterns*. O'Reilly Media, Inc., Sebastopol, CA, 2006.

[10] C. Petrie. It's the Programming, Stupid. *IEEE Internet Computing*, 10(4):96, 95, 2006.

[11] E. S. Raymond. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Sebastopol, CA, Feb. 2001.

[12] V. Subramaniam and A. Hunt. *Practices of an Agile Developer*. Pragmatic Bookshelf, Raleigh, NC, 2006.

[13] D. Thomas and D. H. Hansson. *Agile Web Development with Rails*. The Pragmatic Programmers, Raleigh, NC, 2007.

[14] Tim O'Reilly. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. http://www.oreillynet.com/lpt/a/6228, 2005.

[15] S. Vinoski. WS-Nonexistent Standards. *IEEE Internet Computing*, 8(6):94–96, 2004.