TECHNICAL PURSUIT INC.

APPLIED SOFTWARE ARCHITECTURE

The Keys To TIBET™

EFFICIENT WEB ARCHITECTURES AND AUTHORING MODELS

Twenty-Plus Years In The Web

The web has been around for over twenty years.

The World Wide Web has been in existence for over twenty years, the first versions of "the web" having emerged in late 1990[1].

Web scholars generally agree the 1993 release of the Mosaic browser marked an early turning point in the evolution and adoption of web technology by adding support for viewing images inline with web text.

The Web 1.0 era began roughly around 1995.

By October of 1994 the World Wide Web Consortium (W3C) had been formed and the era of "Web 1.0" had arguably officially begun.

For the next ten years the overwhelming majority of web development focused on authoring text and image content contained in full pages.

Fast-forward to February of 2005.

Web 2.0 was ushered in with the advent of AJAX in roughly 2005.

In a paper entitled "Ajax: A New Approach to Web Applications" author Jesse James Garrett described using asynchronous JavaScript calls and XML to build web applications which didn't rely on full page refreshes.[2]

Use of "Dynamic HTML" or DHTML existed years prior to the coining of "AJAX", but 2005 marked a second turning point. "Web 2.0" had begun.

Which brings us to today.

Still, there are numerous criticisms of web apps, particularly those in the enterprise back office.

Despite all the dramatic progress we've made since 1990 it's all too common for users and web developers alike to voice strong criticisms of web applications, particularly "enterprise" or "back office" applications.

We believe web evolution along two axes is going to provide the solution.

Our work helping companies address common web user and developer issues has led to an awareness there are two key axes along which the web is evolving. More importantly, a web technology's position on these two axes has a direct impact on how it affects end-users and developers.

[1] http://en.wikipedia.org/wiki/World_Wide_Web

[2] http://en.wikipedia.org/wiki/Ajax_(programming)

The Axes Of Efficiency

If we look at the evolutionary differences between Web 1.0 and Web 2.0 one of the first things that jumps out is the shift in architecture.

The first shift from Web 1.0 to Web 2.0 is an architectural one.

Web 1.0 is a predominately a server-centric architecture in which the client device is often nothing more than an HTML rendering surface. Web 2.0 on the other hand is characterized by a much more "involved" client, one that is responsible for more than simply rendering HTML.

A second thing that jumps out between Web 1.0 and Web 2.0 is the shift in authoring technology.

The second thing that jumps out is how the authoring model changes.

With Web 1.0 the overwhelming emphasis was on authoring via tags, in particular HTML tags, augmented with CSS for styling. In Web 2.0 there's a strong emphasis on JavaScript. HTML and CSS remain foundational, but writing JavaScript is a large part of Web 2.0.

So we have two key axes of change: architecture and authoring. The question is how should these be ordered? And what else is on them?

We see Architecture and Authoring as the key axes of web efficiency.

Let's start to answer those questions with a deeper look at architecture.

Architecture

Perhaps the most important choice you'll make regarding your web applications is "What architectural model will you use"?

Your choice of web architecture is your most important choice.

The load has to be carried; but you decide where.
To create compelling change, change your architecture.

When it comes to web architecture it's important to remember that in some sense the workload is the workload, the tasks your application is designed to support have to be done. You choose how and where.

As you can imagine, your choices on how and where the work will be done are central to how efficient and effective your application will be.

There are five layers that define our architecture:

Interaction,

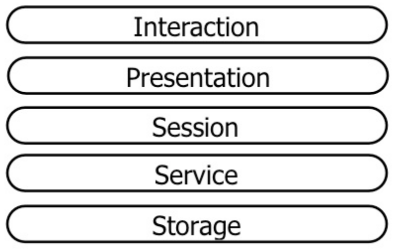
Presentation,

Session,

Service,

Storage.

Web applications effectively have five key layers: Interaction, Presentation, Session, Service, and Storage:



If you think about how these layers interact when you're running offline its clear that it's possible for all five layers to run in a client device. It's not, on the other hand, possible for all five to run on the server.

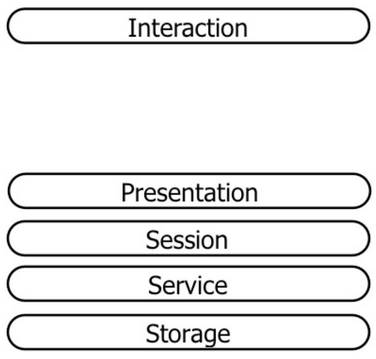
By moving layers, or portions of layers, between the client and server we create architectural variations, each with different efficiency profiles.

Although there are, in some sense, an infinite number of variations to be created we see five particularly relevant ones for most web applications.

Architecture: Web 1.0

Web 1.0: Only the Interaction layer was in the client. Still true for many mobile sites.

The first variation is Web 1.0 or "thin client". It's what you've probably been living with up until the advent of AJAX and Web 2.0:



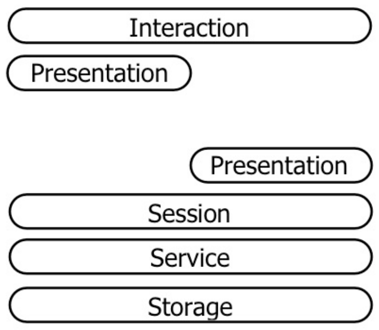
In this variation only the interaction layer, the layer responding directly to user mouse clicks and keys, is in the client. All other work is performed on the server. Network and server loads are at their maximum, user delays are high, and scalability and fault tolerance are at their worst.

Which explains why everyone's looking at Web 2.0....

Architecture: Web 2.0

With Web 2.0 you shift a portion of the presentation layer to the client.

Web 2.0: Interaction and some Presentation now in the client.



There's a little less server overhead, a smaller network footprint, and improved user throughput thanks to fewer server round-trip delays.

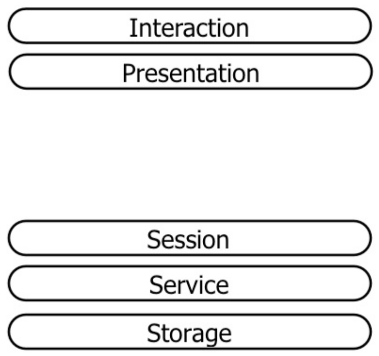
AJAX, but hopefully not badly. It's easy to find that you've added overhead, not removed it.

Note, your server is still generating portions of your UI, your network is still transporting static presentation elements, and overall server hits likely haven't changed significantly -- they may have even gotten worse. Scalability and fault tolerance aren't typically improved over Web 1.0.

Architecture: Web 2.1

With a bit more effort you can take the next step and move the remaining presentation elements to the client.

Web 2.1: Interaction and all Presentation in the client.



More aggressive Web 2.0 efforts often fall into this category, which is why we label it Web 2.1, a variation on Web 2.0, but an important one.

Templating is now done entirely client-side. Yes!

Web applications commonly use "templates" containing the static elements and placeholders such as {foo} for the dynamic elements.

In 1.0 and 2.0 designs the data and template are merged on the server.

A more efficient model is to cache the templates in the client and merge the data there. The result is that the static portions of your application only move once -- and since they're static they take full advantage of the web's tendency to cache content. Only your dynamic data is on the wire, improving overall server load and network load significantly.

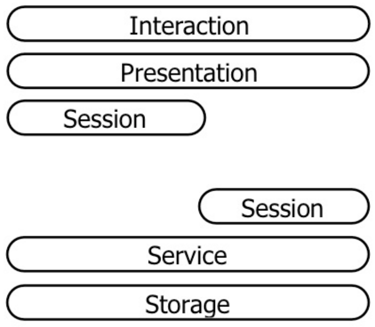
Architecture: Web 2.5

Web 2.5: Interaction, Presentation, and part of the Session are now in the client.

Having eliminated the overhead of moving the same template data repeatedly you can turn your attention to the other performance sink in web applications -- session management.

Session management is a response to the fact that HTTP, the protocol that drives the web, is stateless. This causes problems with sequences of activity that needed to share data across page refreshes when you're using an architecture that relies on refreshing the page.

With AJAX changing the granularity of your server calls the page isn't always refreshed. With proper use of AJAX at least some of your "session context" can be maintained by the client between full page refreshes.



Done right this can drop server communication needs significantly.

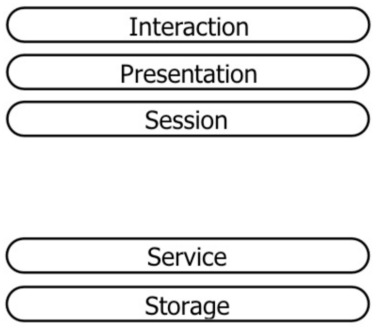
Intelligent analysis of your workflow and session requirements can often allow you to move to a "2.5" architecture in which responsibility for session data is shared between client and server.

With shared session management you further reduce the number of times you need to access the server, further improving scalability and opening the door to certain limited forms of fault tolerance.

Architecture: Client/SOA

The last step in our architectural evolution is what we call Client/SOA -- an architecture that is pure client/server for the web.

Client/SOA: Interaction, Presentation, and all Session (other than Authentication) are in the client.



How is this possible?

If you recall, HTTP is stateless -- in other words -- HTTP calls are atomic.

If your REST API is atomic, meaning clients send transactionally atomic requests, you can eliminate server-side session management completely.

(NOTE that we differentiate between session-based state management and Authentication. Authentication is something you can manage a number of ways without the overhead of a full server-side session).

The application can now run entirely offline provided it leverages browser storage correctly.

You may already doing this if you're doing SOA at the B2B level. Just imagine your web application is just another SOA client and you've completed the journey to the most efficient web architecture you can achieve.

In a Client/SOA model not only is the server able to focus purely on SOA tasks, you've eliminated a common area of development duplication by avoiding the need to either repackage data for different clients or support different APIs for SOA vs. your desktop or mobile applications.

You can use a single set of APIs to interface to backend systems, one of which can be a simple authentication server.

Architectural Summary

As we mentioned earlier, yes, it is possible to move the other layers of the stack to the client, that's what happens when a user runs offline, provided your applications are built to support offline operation.

Let's take a quick look at our architectural variations in summary form.

With respect to architecture we believe there are five important metrics that help us gauge overall efficiency: server load, network load, user throughput, scalability, and fault tolerance.

As you can see as as you move from Web 1.0 to a Client/SOA model you get incremental improvements across all five metrics:

Here they all are, with their relative scores in terms of load, scalability, etc.

	1.0	2.0	2.1	2.5	Client/SOA
Server Load	poor	fair	good	good	best
Network Load	poor	fair	good	good	best
User Throughput	poor	fair	good	good	best
Scalability	poor	poor	fair	good	best
Fault Tolerance	poor	poor	poor	fair	good

Authoring

We see five specific authoring choices composed of varying combinations of markup and AJAX.

From an authoring perspective we believe there are five technology choices: HTML5, HTML5+AJAX, vendor-specific XML such as XUL or Flex, W3C XML such as SVG, XHTML, XBRL, etc. , and W3C XML+AJAX.

As with architecture, we wanted to score each of these options using metrics we felt were relevant to building Enterprise web applications.

We feel the key metrics regarding authoring technology are: available talent pool, tooling, abstraction level, extensibility, and B2B integration.

Obviously there's room for argument in any set of metrics, but we feel this set highlights the most important trade-offs between the options.

Authoring: HTML

This one's relatively easy. HTML and CSS are the core technologies of web development. While you can argue about Java vs. Ruby vs. Perl vs. Python you're eventually going to end up authoring HTML and CSS.

HTML/CSS is easy to do and easy to hire for, but lacks many application support features.

The good news is that they're relatively easy to hire for and there are good tools for constructing HTML/CSS pages.

The bad news is HTML doesn't offer application-level abstractions, isn't particularly extensible, and doesn't offer much for service integration.

Authoring: HTML/AJAX

Adding AJAX to HTML unfortunately takes you in two directions at once.

Adding AJAX helps, but also can create trouble if you hire inexperienced developers, or can't find them at all.

AJAX allows you to extend the capabilities of HTML by simulating application-level components via scripting. As a result it opens up fair abstraction and extensibility options.

Unfortunately hiring for AJAX is a challenge, few libraries are "Enterprise class", and the tools supporting JavaScript remain fairly immature.

Authoring: Vendor-specific Markup

There are several options for vendor-specific or, at the least, non-W3C endorsed markup. The tradeoffs you make with this choice are pretty well understood.

Vendor-specific markup options such as FLEX are another choice, but can be dangerous.

Choosing to go with a proprietary technology quite often improves both the level of application abstraction and the tooling for developers.

Depending on the technology your available hiring pool may be smaller and your ability to extend and/or integrate may be limited. But with the right vendor it could easily go the other way. Our industry has often grown in the direction of a dominant vendor-backed technology.

The critical downside? Vendor lock-in.

Beware of lock-in.

Authoring: W3C/* Markup

W3C-standard XML is another option. Based on your industry you may already be using this.

Choosing to go with an approved open standard from the W3C is, like some of the other options here, a bit of a double-edged sword.

As a specific example, XForms was, at one time, touted as the future of web forms. In retrospect it's clear that XForms is almost completely marginalized and that HTML5 (or XHTML5 if you prefer) is the future.

Still, at least XHTML5 is a standard, not a vendor-specific technology. The point is you have to choose your standards wisely. Not all of them end up finding sufficient support to carry them across the chasm.

In the short term the talent pool for some of these technologies is small, and tooling ranges from non-existent to fair depending on the markup.

Interoperability is a big upside feature here.

Where these technologies can often excel is in their level of authoring abstraction and their ability to foster standards-based service integration.

Authoring: W3C Markup w/AJAX

One notable downside to W3C standards is that they typically lag market realities and can become a form of “standardized straightjacket”.

You’ll probably need to augment with AJAX to avoid falling behind the industry.... Standards bodies move slowly.

For example, XForms doesn’t offer any way to bind to non-XML data but many web services are being made available in JSON, a JavaScript object format. So XForms fails the real-world test imposed by JSON.

As with HTML however, many limitations in the W3C standards can be addressed by combining W3C-standard XML with AJAX -- provided you’re using a platform that supports such integration and extension.

Authoring: Summary

To summarize, while the choice of authoring technology is a little more open to debate we think there’s a solid case to make XML authoring, particularly with W3C, IETF, and OASIS standards, augmented by AJAX.

If we look at our options and their relative scores we see that there's no clear winner, no option we'd consider "best" by a clear margin:

Authoring model is less clear cut in terms of an obvious "winner".

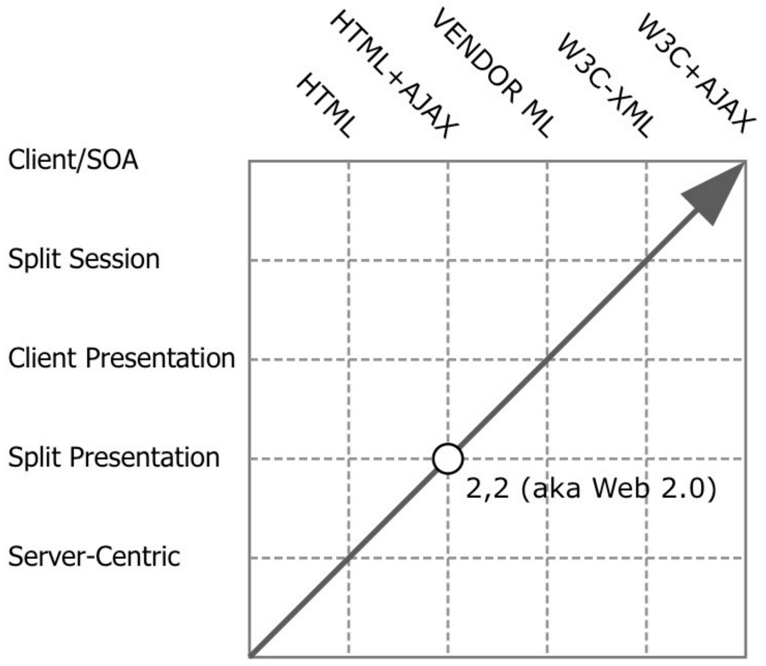
	HTML	+AJAX	VEND.	W3C	W3C+A
Talent Pool	good	fair	poor	fair	fair
Tooling	good	fair	good	good	fair
Abstraction	poor	fair	good	good	good
Extension	poor	fair	fair	fair	good
Integration	poor	poor	poor	good	good

Still, if we consider the overall trend from the past several years, it does seem that as requirements for web applications become more complex the balance shifts toward well-tooled W3C XML augmented with AJAX.

We believe that over time XML+AJAX with good tooling will "win".

Putting It Together

So where does that leave us? Here's how we see it:



If you put it all together you get a sense of where the web is today with "Web 2.0" and where the web could be as we evolve toward semantic markup.

In our summary diagram you can see the architectures on the Y axis and the authoring models on the X axis. Web 2.0 falls around 2,2 on the grid.

We see a lot of room for improvement.

You can plot pretty much any technology choice on the graph to see how it measures up and what trade-offs it might imply.

The nice thing about this diagram is you can plot most web technologies on it somewhere, giving you a quick visual way to compare and contrast.

For example, a technology that shares responsibility for UI generation between client and server (split presentation) and which you author in HTML/AJAX would come in around the 2,2 mark.

As another example, there are server-side XForms products with/without AJAX capability. These would be found at roughly 5,2 on the grid. More powerful abstractions to be sure, but limited in terms of performance, scalability, and fault tolerance based on our scoring parameters.

We believe the industry is heading for 5,5 on the graph....

Our sense is that, as an industry, we're slowly working our way to using technologies we'd characterize as 5,5; technologies that let us author in standardized XML+JS while leveraging a Client/SOA architectural model.

Summary

We clearly have our own ideas about efficiency, architecture, authoring, and the evolutionary direction of web development.

Where you are on the graph, and where you should be, depends on each application's requirements.

What choice is right for you? That depends on what your application needs to do, who its users are, what kind of devices are they running, what kind of developers you are able to hire and retain, and so on.

Our goal is simply to present one way of visualizing your choices and their relative strengths and weaknesses. There are always other ways.

There's no "one true answer" that works for all applications.

Contact us for more at:

info@technicalpursuit.com

Contact us for more information, or to discuss how we can assist you in making your web applications more efficient and productive.