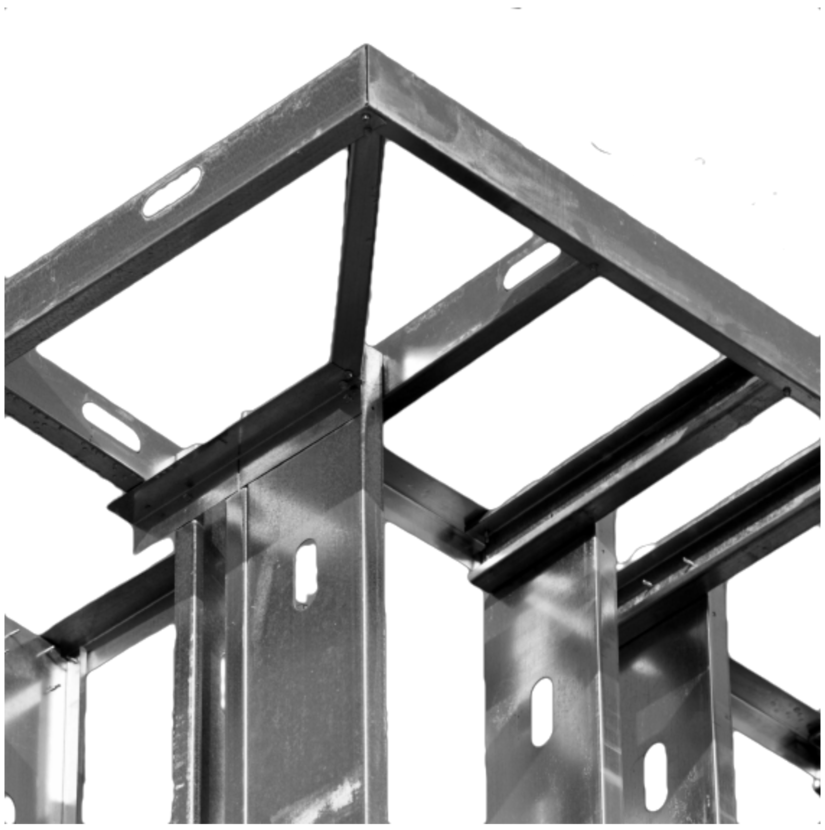
T E C H N I C A L   P U R S U I T   I N C .

W E   W R I T E   J A V A S C R I P T   S O   Y O U   D O N ' T   H A V E   T O . ™

The Axes Of TIBET™

COMPARING WEB ARCHITECTURES AND AUTHORING MODELS



Twenty-Five Years In The Web

The web has been around for twenty-five years.

The World Wide Web has been in existence for roughly twenty-five 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.

Fast-forward 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 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 stronger emphasis on client-side JavaScript. HTML and CSS remain foundational, but coding in 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 workload 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,

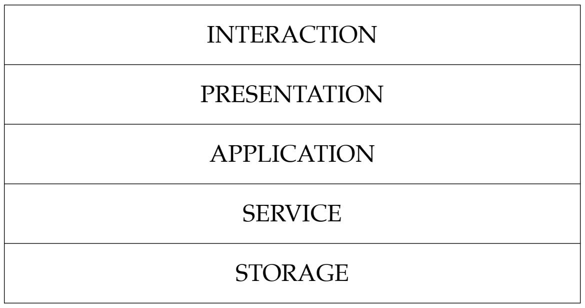
Presentation,

Application,

Service,

Storage.

For the purposes of our discussion web applications have five common layers we can chose to migrate across the client/server boundary:



The interaction layer handles user input via keyboard, mouse, or other input devices. Presentation handles UI generation. The application layer handles logical flow in, and between, screens. The service layer isolates our application logic from specific service dependencies. The storage layer ensures data is persistent across sessions as needed.

How we arrange these layers between client and server is key.

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

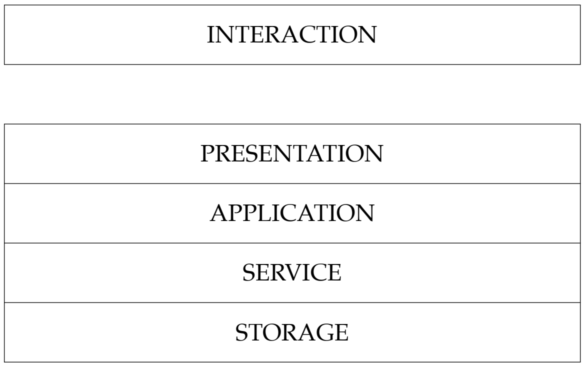
We see five particularly relevant alternatives for most web applications.

Architecture: Web 1.0

Web 1.0:

Only the Interaction layer is in the client.

The first variation is Web 1.0 or "thin client". It's what most web sites and applications relied upon 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.

We focus on 3 common variants of Web 2.0.

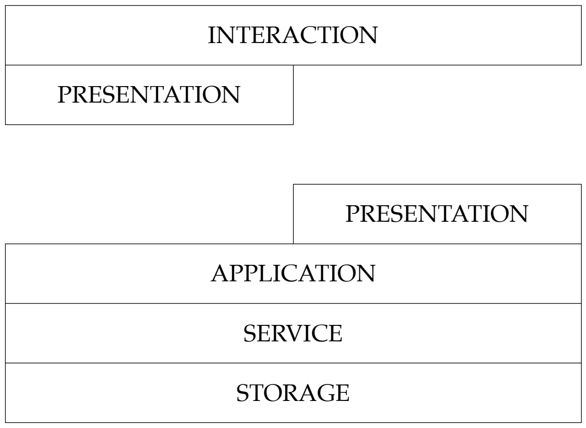
Unfortunately "Web 2.0" can cover a wide range of different alternatives in terms of architecture. We've chosen to focus on 3 common variations.

Architecture: Web 2.0

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

Faster response times on a very localized basis, but still dependent on both server and network.

With Web 2.0 you shift a portion of the presentation layer to the client. This is the most predominant variation for web sites vs. applications.



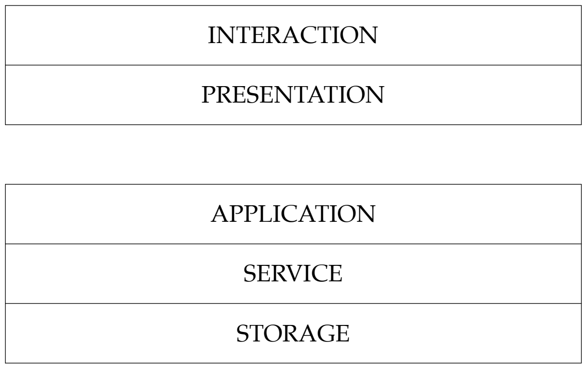
The server is still generating portions of your UI and your network is still transporting static presentation elements. Page-specific latency may be better but there's still a strong dependency on both server and network.

Architecture: Web 2.1

Web 2.1: Interaction and all Presentation in the client.

Faster rendering and less static data on the wire. Still fully-dependent on server and network.

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



Separation of static and dynamic portions of data improves caching and overall performance.

By pushing page templates, which are static, to the client and merging the dynamic data in the client you can leverage browser caching more efficiently. With static vs. dynamic content separated properly the overall server load and network load are reduced considerably.

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.

Architecture: Web 2.5

Web 2.5:

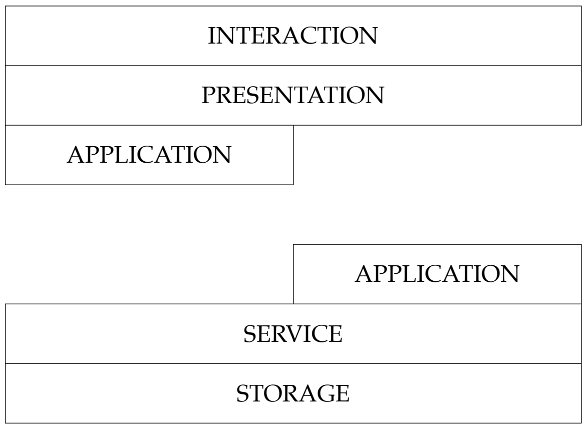
Interaction, Presentation, and some Application logic is now in the client.

Having eliminated the overhead of repeatedly sending the same static template text across the wire you can turn your attention to the other common performance sink in web applications -- session management.

HTTP, the protocol underlying all web applications, is stateless. By itself that's not a problem. But if you keep all your controller logic on the server you create architectural tension since you now need a way for the server and client to stay in sync. You suddenly need "session data".

Additional performance improvements to specific use cases. Still reliant on some measure of "session management."

By avoiding full page refreshes and working more granularly it's possible to greatly reduce the need for state management via sessions. Most so-called "single-page" applications migrate at least some application logic to the client:



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 which provides client-side logic for your user's most common use cases.

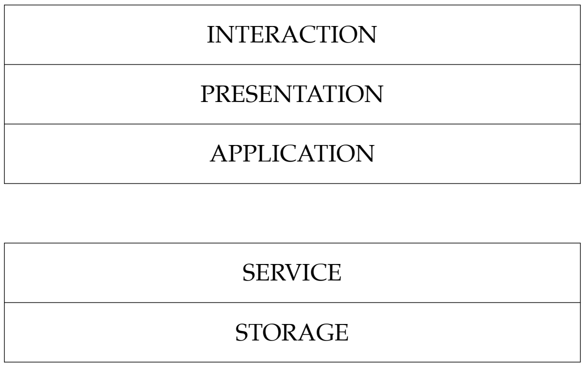
By reducing the need for session management you reduce the number of times you need to access the server, further improving scalability and opening the door to localized areas of increased fault tolerance.

Architecture: Client/SOA (aka RESTful)

Client/SOA:

Interaction, Presentation, and all Application layer logic in the client.

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



If you recall, HTTP is stateless. Essentially all HTTP calls are atomic.

Leveraging a pure REST API on your server lets you achieve this model.

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

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

Going Offline

By leveraging modern application caches an application using a Client/SOA model can often be run offline. Or you can go one step further....

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

Most well-factored application layers will include some form of service interface. As a result, it's possible to migrate client-specific versions of the last two layers to the client, creating applications which don't rely on a server at all; applications which sync their data stores when online.

TIBET+CouchDB goes further and lets you run offline and sync online.

TIBET on CouchDB is an example of this offline-ready architecture.

The key to designing your architecture is knowing your application layers and distributing them to meet your needs.

Architectural Summary

As we mentioned earlier the key to understanding and then optimizing your architectural choices is to understand the layers of your application so you can make conscious decisions about how to distribute them.

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:

For the variations we've outlined here are their relative scores in terms of load, scalability, etc.

METRIC	1.0	2.0	2.1	2.5	C/S
SERVER HITS	poor	fair	good	good	best
NETWORK	poor	fair	good	good	best
USER LATENCY	poor	fair	good	good	best
SCALABILITY	poor	poor	fair	good	best
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 (XUL, Flex), W3C XML such as SVG, XHTML5, 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.

While the choices here are somewhat more subjective we feel the key metrics regarding any authoring technology choice are: available talent, tooling support, 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: HTML5

This one’s relatively easy. HTML5 and CSS are the core technologies of modern web development. While you can argue about Java vs. Ruby vs. Perl vs. Python vs. Node.js on the server you’re eventually going to end up authoring HTML5 and CSS to get a browser to render your app.

HTML5/CSS is relatively easy to do and relatively easy to hire for...

The good news is that HTML5 and CSS are easier to hire for than JavaScript and there are tools for authoring with these technologies.

...HTML5 unfortunately lacks many application support features.

The bad news is HTML5 doesn’t offer application-level abstractions, isn’t particularly extensible, and doesn’t offer much for service integration.

Authoring: HTML5 + AJAX

Adding AJAX to HTML5 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 developers on schedule.

JavaScript allows you to extend the capabilities of HTML5 by simulating application-level components via scripting. As a result it opens up fair abstraction and extensibility options at the expense of some complexity.

Unfortunately hiring becomes a challenge, few libraries are "Enterprise Class", and the tools supporting JavaScript remain relatively immature.

Authoring: Vendor-specific Markup

There are several options for vendor-specific or, at the least, non-W3C markup including XUL and Flex. The tradeoffs with this choice are pretty well understood.

Vendor-specific markup is another option. There are a number of choices here.

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

Abstraction level and tooling can often be better...but you may get some level of lock-in.

Depending on the technology your hiring pool may actually be smaller and your ability to extend and/or integrate custom code may be limited.

Authoring: W3C XML

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

Choosing to go with an approved XML 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) is "the future"...for now.

Still, at least XHTML5 is a standard, not a vendor-specific technology, and XHTML5 has the benefit of being XML so you don't have to abandon tools like XML Schema or XSLT as you do if you use HTML5 instead.

Interoperability is a big upside feature here.

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

Authoring: W3C XML + 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 didn't offer any way to bind to non-XML data but the modern web is heavily JSON-oriented. XForms further didn't provide for extension via JavaScript, hence XForms fails the real-world test imposed by JSON-based data sources and the need for extensibility.

As with HTML however, many limitations in the W3C standards can be addressed by combining W3C-standard XML with AJAX.

Authoring: Summary

There's no clear winner, but we believe in basing things on W3C XML extended via JavaScript.

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

While XML is not without its issues, the tooling available for XML can often be exceptional and there are complementary standards like XML Schema and XSLT which can make certain tasks both easier and faster.

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:

As our summary chart shows authoring can be a very subjective topic and there's no single "best" approach that jumps out.

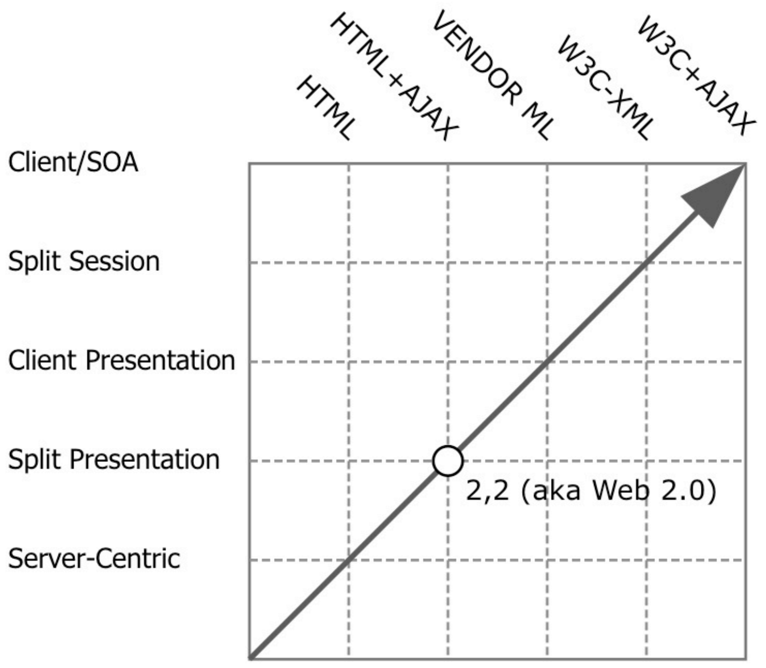
METRIC	HTML5	+AJAX	VEND.	W3C	+AJAX
TALENT POOL	good	fair	poor	fair	fair
TOOLING	good	fair	good	good	fair
ABSTRACTION	poor	fair	good	good	good
EXTENSION	poor	fair	fair	fair	good
INTEGRATION	poor	poor	poor	good	good

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 prevail for Enterprise applications.

Putting It Together

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



The graph here helps to put things in perspective.

The industry has largely evolved from server-generated HTML to Web 2.0. We see the trend continuing up and to the right until we reach a true client/server model.

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

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

TIBET is built to hit the 5,5 mark on the graph.

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

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

What matters is that you choose consciously (and wisely) for your needs.

Contact us for more at:

info@technicalpursuit.com

The nice thing about our "Web 5x5" diagram is you can plot most web technologies on it giving you a quick 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.

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+JavaScript while leveraging a Client/SOA architecture.

That's the point on the graph TIBET is designed for, Web 5.5 as it were.

Summary

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

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. The key is to choose consciously and wisely for your application needs.

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