Orchestrating Web Engagement Requires Dynamic Context Intelligence

An optimized web experience means putting the right content on the screen first in the least amount of time — given the user's browser, device, location, and likely behavior.

The problem with conventional web optimization approaches is that they just speed up loading the whole page at once. What is wrong with this approach? The problem is that users do not care about all content on a page equally. Graphic designers have known for years that where users see information displayed on a page matters. So it makes sense that *when* users see information displayed also matters — as in how fast they get to see certain content and in what sequence. So if you wait to display *anything* before you can display *everything*, you are not giving users what they want when they want it. They will have to wait for the content they care about less in order see the content they care about more.

An alternative to the whole-page-fast approach is context-based orchestration. In that model *what* users want to see drives *when* they see it. And that what/when decision depends on a whole host of factors that together make up the context, including the user's browser, device, geographic location, and likely behavior in response to the content. The idea is that by delivering what users actually want rather than what a site owner thinks is important, you'll see a boost in metrics like bounce rate, click through, and session time.

If the premise of context-based orchestration is true then it begs the question: how can it be implemented? One possible approach is a rulebook for web developers. This might not be practical because of how fast the web keeps changing: some optimizations that worked yesterday do not work as well today — domain sharding, for example. Moreover, context-based orchestration is highly infrastructure-dependent, hence outside the control of most web developers. Organizations, however, don't have to build orchestration technology *into* their website in order to apply it to their website. They can focus on creating compelling digital experiences and outsource delivery optimization to the cloud.

What site owners *will* want to know is how context-based optimizations offer businesses advantages over competitors that do not apply them. The secret sauce is a cloud-based object model of their website — so the actual website does not need to change in order to change how content arrives. Applications apply rules that transform the model into an optimized proxy of the original website using a configuration language consistent (and hence scalable) across all models. Everything about the model can be decided by applying these rules, including where geographically content is cached for faster loading. Decisions are informed by integrated analytics and by data from browser agents regarding user actions and content load times — creating a feedback loop of content delivery to continuously prioritize whichever specific pieces of content engage most.

> *"Mobile devices now generate over half of all web traffic, so many website owners have learned they need to design their sites to fit both the mobile device and the mobile user."*

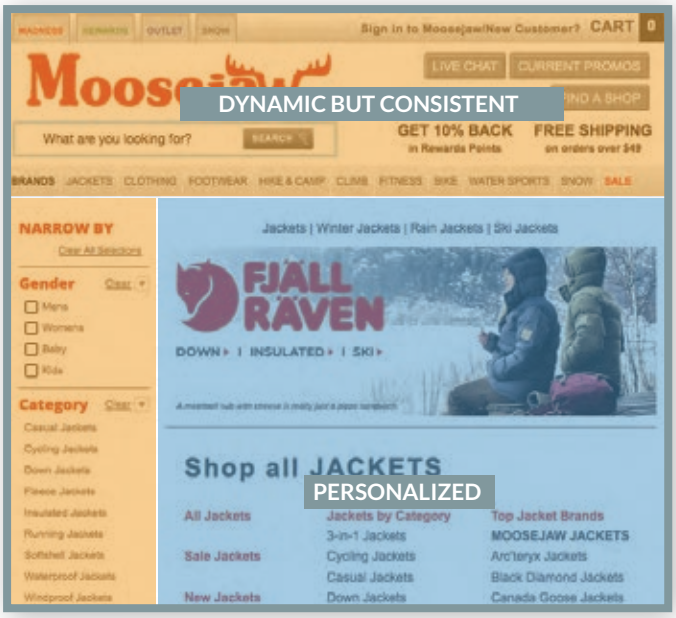
The Whole Page Fallacy

In an ideal world a whole web page would always load instantaneously. But in the real world, trying to load whole pages at once is not ideal, given these six key insights:

1. Most Web Resources are Static

In most web applications users need to look at and absorb some static content before they can begin the task they've come to do. As such, most web app pages are still comprised primarily of static content, with dynamic content (a cart checkout button, for example) representing a tiny share of the overall page. Because static content, by definition, is much less likely to change than dynamic content, it is much more likely to be available in cache, and probably exists in a cache close to the user via a content delivery network (CDN).

The result of all this is that loading all the static content first makes the user feel engaged. They don't miss what they're not looking for yet — the dynamic content. What they perceive initially is what appears to be a full page of content: and if caching is properly leveraged, the page seems to display instantly. If one is seeking to achieve a high-performing website that takes 1000 milliseconds to display all page content, and it requires only 200 milliseconds to display the static content, that leaves a budget of 800 milliseconds left over in which to fetch and display the few pieces of dynamic content.



Even when page elements are dynamically served, they are not personalized to the user and can be cached.

2. Whole Page Loads Intrinsically Degrade Engagement.

If you ever view a web page from a local server you might be surprised at how slow the page loads. The fact is, most websites display noticeably slower compared to, for example, pages already cached in your browser — *even when no network contention or distance gets in the way*. A delay that still exists when there's no Internet between server and browser is not a delay that a CDN can remediate. The whole premise of a CDN is to speed up page loads by moving content closer to the user.

There are a number of reasons pages load slower than they could, whether accessed locally or over the Internet. Some have to do with how a web page is coded, others with the web infrastructure itself. Coding factors would include things like whether there are delays in JavaScript or CSS execution that could be avoided — for example, by concatenating smaller blocks of code into bigger blocks or by removing bottlenecks, such as when some JavaScript has to wait for some CSS or vice versa.

Infrastructure factors would include things like cache density. To speed up page loads, most modern websites add cache to their web infrastructures, such as when they employ CDNs or Varnish, a server-side open source system for caching user requests. Most of the content users see on their screens actually comes from a cache and there are typically several layers of caching between the user and the origin server. Cache density refers to how much of a website's content can be served from one cache. The more of the page's content that's cacheable the denser the cache, and the faster content loads. That's because each separate cache a browser needs to access in order to completely load a page, or to load multiple pages of the same site, requires a separate call to a separate web address.

These are challenges built into most websites and most web service infrastructures that make the decision to load whole pages at once intrinsically degrading to engagement.

3. Web Users are Application-centric, Not Page-centric

We are long past the days when most websites were just online versions of brochures. Even books and newspapers have become highly engaging online vehicles for user interaction. Websites have become true applications the client executes in order to complete tasks the user wishes to accomplish — like purchasing products, banking, and game playing — among thousands of examples that could be cited. When the *purpose* of web content shifts from page display to task performance, optimizing for page display loses its purpose. How well you engage gets redefined as how well you make the user *feel* enabled. In that context whether all the content gets displayed at once is beside the point. What matters is having enough of the right content displayed so the user doesn't feel like he or she is waiting for a gate to open in order to get something done.

4. How to Engage Best Depends on the User's Device

Mobile devices now generate over half of all web traffic, so many website owners have learned they need to design their sites to fit both the mobile device and the mobile user. Compared to desktop PCs, mobile devices typically have smaller screens, slower connection speeds, and a soft interface rather than a keyboard and mouse. Mobile users are also much more likely to be using a website while busy with other activities like mall shopping. That's why, for example, there are responsive websites — i.e., websites that change window size, layout, and menu design dynamically depending on screen size.

But how and where content is displayed on-screen is not the only variable to be optimized based on "device context." When content is displayed it must also be optimized to accommodate lower link speeds and cellular networks. Responsive designs illustrate this dichotomy best since the way they optimize a website's appearance for mobile may actually increase load times. That's because they load the whole page, even parts that won't appear on the screen until the user scrolls or taps a menu button. So before users see any content they must endure delays resulting from the loading of content they may never need. And if they are loading this content over a cellular network, that means they are also paying for this unseen content. Optimizing for mobile would avoid this possibility and delay loading content until it is actually requested.

But what is true for mobile users also applies to desktop users:

5. Waiting for Hidden Content is not Optimal

Unseen or hidden content also degrades the desktop user experience as well. Hidden content is content on a page the user must click on to see, such as product reviews or FAQs, where the user's action expands the page to reveal the content. In a non-optimized website the user may experience additional slowness and a "choppy" user experience until the hidden content is loaded "behind the scenes." In a context-optimized website, the hidden content could load after an intentional delay so that it does not interrupt the rendering of visible content.

3rd party content like this Twitter widget can improve user engagement unless it slows page load time and creates bottlenecks in the user experience.

6. Web is Changing Very Fast

Finally, another reason to not base optimizations on whole pages is that it force fits a static model onto a dynamic environment. New web technologies and use cases (think mobile and responsive design) are continually creating unforeseen contexts that conflict with existing assumptions about what experience best engages different users. The only way to match the page loading experience to the user is by being able to adapt how the page loads, granularly and regardless of how the web evolves or how quickly.

That ability to adapt requires context intelligence, which optimizing only for fast page loads doesn't offer. This requires intelligence in the areas of both analytics and rules. In other words, you need to know what context you are optimizing for and then have the ability to do something about it.

Gathering this knowledge is only possible if there is real-time feedback about what parts of a page engage users most. For example, will the level of engagement change if some content displays before other content, or if users don't have to scroll to see certain content?

But this information isn't useful if you can't do something with it — that is, if you can't actually change how content actually loads given certain conditions. That means you need rules, an engine to apply them, and a rules architecture. You need to be able to layer those rules on top of each other in a coherent way so that you're not constantly reinventing the wheel with every rule you add.

The Whole Page Alternative — Context Intelligence

The key difference between whole-page versus context intelligence is optimized orchestration — the ability to choreograph how pieces of content are delivered for maximum engagement, taking into account the context in terms of technology and use case. So what would context-based orchestration look like? What would be the key strategies and what key building blocks would best enable those strategies?

Clearly, the strategies would exploit the six insights just discussed, as in:

1. No Website Modification

Forcing developers to change how they code websites flies in the face of the dynamic web. Developers are tasked with providing compelling digital environments and rely on the infrastructure to deliver them. Asking developers to also continually re-code how sites are written so that the right content is optimally delivered based on changing technologies and use cases is impractical. So, if context-based orchestration is going to work it's got to be hands-off the website.

2. Analytics and Rules

As just discussed, intelligence is all about having knowledge of what actually engages users based on their actual behavior on the site, and rules to determine how content is delivered in a manner aligned with that behavior.

3. Layered Optimization Architecture

If you are going to have rules, then you also need a practical way to apply them within the content delivery infrastructure. Just as you wouldn't want developers to have to continually recode their sites to stay current with the changing web, you wouldn't want to continually retool the underlying infrastructure in order to employ the most advantageous rules for orchestrating content delivery. That suggests rule-based optimizations that are layered so they could be added or subtracted incrementally and can build off each other logically.

4. A Document Object Model

If you are going to have rule-based optimizations to change how content is delivered, it invites the question, what exactly will these optimizations be changing, especially since the website is "hands off"? Generally, there are only two ways to optimize how content is delivered: change the website code itself or change the content as it passes through the infrastructure (via a CDN, for example). But if the website is off limits, that means that you need to create a proxy, or document object model, of the site and apply the site-related optimizations to the model.

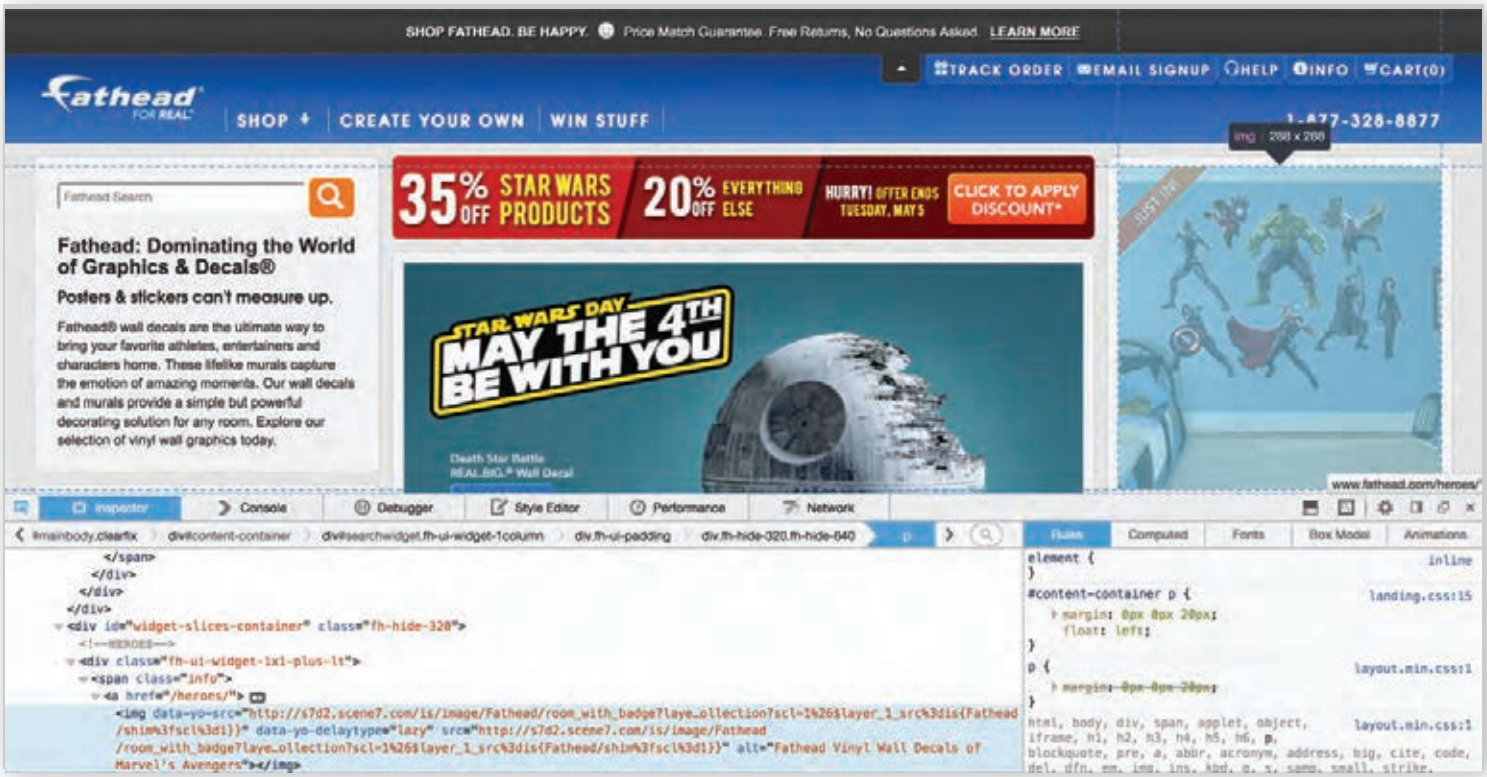
5. Consolidate Cache Keys for Denser Cache

On the infrastructure side, one of the optimizations that can be applied is to redirect all requests to a given origin server to the same cache, regardless of the requested URL. This would mitigate the problem that arises when the same content is cached in multiple locations because different URLs were generated for the same content (as happens in a Google search result, for example). Eliminating these extraneous caches means that a user's browser makes only one call to get the same content, greatly reducing load time.

6. Load Static Content Right Away

Here's an example of a website optimization performed via applications on the object model, which also doubles as an infrastructure optimization. At the model level, the optimization is to *display* static content first. That fills up most of the page instantly, making the user feel immediately enabled. At the infrastructure level, the optimization is to deliver content in closest geographic proximity to the user, which is likely to be static content. That makes it even more likely the page will appear filled right away.

> *"The key difference between whole-page versus context intelligence is optimized orchestration — the ability to choreograph how pieces of content are delivered for maximum engagement, taking into account the context in terms of technology and use case."*



*Yottaa's **InstantOn** feature accelerates display of normally non-cacheable static content (like HTML body fragments) and loads it instantly.*

7. Control the browser

How, when and which content gets loaded on the page is ultimately the result of browser actions. So, for example, if you want to load static content ahead of dynamic content, you can tell the browser that is what you want to do. As just discussed, by controlling the object model proxy you can control the sequence of which kind of content (code) gets rendered (executed) — even if that's a different sequence than in the origin website — thus giving static content priority. You can also optimize caching so that content is sourced from the closest cache location — giving static content priority by default.

But browser control also presents other opportunities for context-based orchestration. That would be to *discover* what the user is doing with the browser and feed that information back to the applications controlling the object model and delivery infrastructure. That feedback enables machine learning — i.e., the logic responsible for content sequencing is fine-tuned to better ensure the sequence is based on content the user actually wants to see. Content the user can't see yet, such as an eCommerce security seal at the bottom of the page, might then be delayed until the user actually scrolls to where the content should be visible.

Yottaa's Implementation

Given what we know about user engagement in the era of the application-centric dynamic web and the strategies for context-based orchestration, let's look at Yottaa's real-world implementation as a model. Specifically, what are the key functional modules that actually make these strategies work?

At a high level, the implementation provides five key cloud-based functions: a configuration language, content optimization, traffic management, browser agent, and global load balancing.

Configuration Language

An orchestra requires a conductor. In this case that's a role played by a Yottaa engineer using a configuration language to specify the rules governing how delivery of a website's content should be optimized.
The language is a Java-like syntax that can apply virtually any policy concerning the sequencing of website resources or the behavior of the underlying delivery infrastructure (as in caching, for example).

That includes:

-What CSS and JavaScript code should be concatenated to avoid needless processing delays (like when CSS interrupts JavaScript or vice versa)

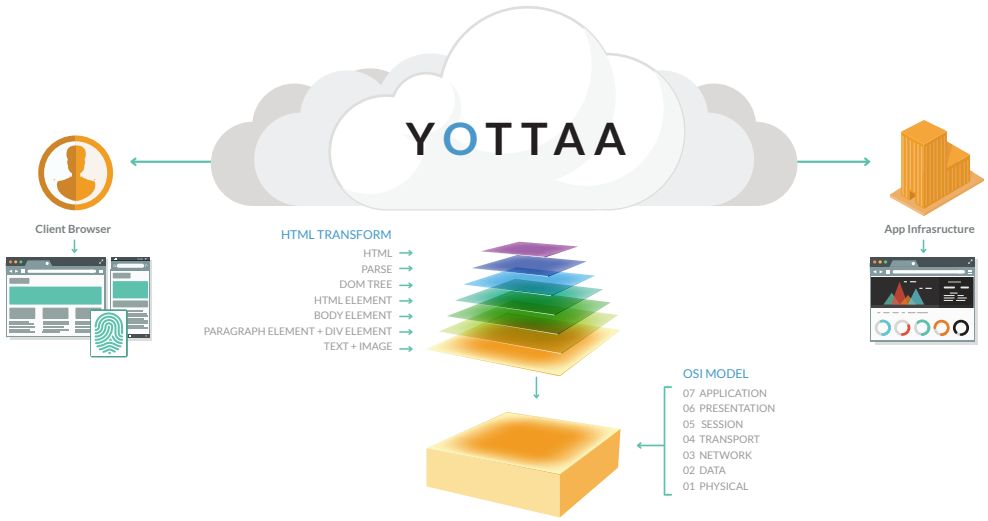
-Where should a particular piece of content be cached, even if different from where other content on the same page is cached

-That a particular piece of content (like a logo) will always be cached regardless of the website's HTML

-Which images, if any, the browser should always pre-fetch

-The maximum number of URLs the browser can connect to at the same time

- Which versions of a piece of content (e.g., the English or Chinese version) are sourced or whether content (e.g., Facebook "like" icons) is sourced at all if there is no reason for the user to see the content (e.g., it links to a prohibited site)



Policies consist of rules that exist in a library as pre-coded applications the engineer invokes using a point-and-click graphical user interface. The result is a rules document that directs how Yottaa modules behave, specifically content optimization, traffic management, and global load balancing. There can even be multiple versions of this document to support A/B testing to see which rule sets (which "what/when" profiles) achieve best engagement.

Content Optimization Module

This module drives on-page content orchestration — i.e., which content gets delivered to the browser in which order. It contains logic that:

-Interprets the rules document

-Executes built-in policies that are turned on by default

-Performs machine learning based on feedback from the browser agent (discussed below) and from integration with third-party analytics so as to optimize orchestration based on actual user behavior

Running the content optimization on "auto pilot" means that only default policies apply and that the Yottaa system will do its best to optimize the site on its own. These default policies represent best-in-class optimizations for most websites and provide a good head start so as to reduce the effort and time needed for further fine-tuning.

Consistent with the layered application strategy discussed earlier, applications invoked by the rules document are layered to easily accommodate new technology and use cases either by adding new layers or deleting old ones. Together these applications and the module's underlying logic comprise context intelligence — the ability to respond to dynamic inputs such as a particular user behavior and website analytics to align content orchestration with real-world conditions.

Traffic Management Module

This module executes applications that determine where specific pieces of content (like a video) are to be sourced and by which route they are to arrive (such as via a specific CDN). It is traffic management, for example, that implements the caching strategy of consolidating all cache tags belonging to the same URL to achieve denser caches and therefore fewer DNS lookups.

Global Load Balancing

Yottaa leverages global cloud services providers such as Amazon Web Services to position its runtime services at geographic locations advantageous for optimum user engagement based on link latency and where users and content are located.

Because those factors may be different for different websites at different times, so may be how Yottaa resources are balanced.

Browser Control Agent

This is code that Yottaa puts in the browser to do two things:

-Tell the context intelligence (via an API) about user actions so machine learning can occur regarding which content is most engaging (e.g., most users run the video first)

-Tell the browser in which sequence to load specific content items in response to the context intelligence

The browser agent essentially closes the feedback loop, creating a virtuous cycle in which initial optimizations improve user engagement, the results of which enable further optimization — and so on.

Hands-Off Optimization

Intelligent context-based content orchestration is also "hands-off" orchestration. Because all services are based either in the cloud or the user's browser, the website itself does not need to be changed. Only its DNS needs to edited so that requests to the website are redirected transparently to the document object model — the website's proxy in the cloud — against which all the optimizations are applied.

That means zero footprint on the business infrastructure and zero implementation overhead, with less than a day to set up, plus a pay-as-you-use service model. Configuration can be done anywhere via the web. And, best of all, results are easily tracked and based on actual engagement metrics (not just technical metrics like latency). That is the sustainable business advantage website owners will want.