



The Missing Mechanic: Behavioral Affordances as the Limiting Factor in Generalizing HTML Controls

Alexander William Petros, Carson Gross, Dillon Shaffer, Matt Revelle

Accessibility Disclaimer:

For a more accessible version of this document, please submit an accessibility request form through the Montana State University Library website.

The Missing Mechanic: Behavioral Affordances as the Limiting Factor in Generalizing HTML Controls

Alexander William Petros*
Montana State University
Bozeman, MT, USA
contact@alexpetros.com

Dillon Shaffer
Montana State University
Bozeman, MT, USA
dillon@molkars.dev

Carson Gross*
Gianforte School of Computing
Montana State University
Bozeman, MT, USA
carson.gross@montana.edu

Matthew Revelle
Montana State University
Bozeman, MT, USA
matthew.revelle@montana.edu

Abstract

In this paper, we analyze a set of three proposals—titled Triptych—which carefully extend HTML to support more generalized hypermedia controls. We evaluate the expressive power of these proposals by demonstrating which user experience patterns they make possible to describe in HTML, and which patterns remain unsupported.

We also introduce the concept of behavioral affordances which characterize common UX patterns in web applications. Through this analysis of UX patterns, we show that HTML currently lacks a native mechanism for expressing behavioral affordances. Finally, we theorize a mechanism for defining arbitrary behavioral affordances that could fill this expressive gap.

CCS Concepts

• **Human-centered computing** → **Hypertext / hypermedia**;
• **Applied computing** → **Hypertext / hypermedia creation**; •
Information systems → *RESTful web services*; **Hypertext languages**; **Web interfaces**.

Keywords

Hypermedia, Hypermedia Controls, HTML, Triptych, Behavioral Affordances

ACM Reference Format:

Alexander William Petros, Carson Gross, Dillon Shaffer, and Matthew Revelle. 2025. The Missing Mechanic: Behavioral Affordances as the Limiting Factor in Generalizing HTML Controls. In *Proceedings of the 36th ACM Conference on Hypertext and Social Media (HT 2025)*, September 15–19, 2025, Chicago, IL, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3720553.3746684>

1 Introduction

The World Wide Web (WWW) is the largest and most significant hypermedia system in the world, eclipsing previous hypermedia in practical relevance almost immediately after its introduction [5]. Its standardized Hypertext Markup Language (HTML) is free to use,

*Both authors contributed equally to this research.



This work is licensed under a Creative Commons Attribution 4.0 International License. *HT 2025, Chicago, IL, USA*

© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1534-1/25/09
<https://doi.org/10.1145/3720553.3746684>

easy to generate, and understood by pre-installed clients on many desktops, laptops, mobile devices, and countless other computers.

The ubiquity of web browsers makes HTML the world’s most useful hypermedia standard, but also a correspondingly conservative one. Backwards-incompatible changes to web browsers not only harm the WWW’s archival properties—by rendering inaccessible websites that lack the resources to upgrade—they also compromise the widespread availability that is the web’s core value proposition to developers. The web requires the developer to surrender control of the client, so it must provide a client that is almost as stable as what the developer might have created for themselves. For these reasons, breaking changes to the standards are highly discouraged and exceedingly rare [1].

Despite the difficulty of doing so, extending HTML is a worthy goal. Semantic HTML is a stable target that can be interpreted not just by standard web browsers, but by screen readers, search engines, and structured data tools. It is a human-readable, declarative interface specification that can be written by programmers and non-programmers alike. Pure HTML, without any style or scripting enhancements, is among the most important data formats on the planet.

In this paper, we consider: what does it take to extend the semantics of the world’s most significant hypermedia system, where clients number in the billions and the vendors themselves cannot guarantee updates? Where hypermedia has, in short, become truly “feral” [30]. To answer this question, we analyze Triptych¹, an in-progress attempt to extend HTML to support generalized hypermedia controls of the kind described by Gross et al. [18]. We contrast Triptych with implementations of generalized hypermedia controls commonly found in modern web pages. This analysis reveals what functionality would still remain unsupported in Triptych-enhanced HTML and which cannot be easily added to HTML while preserving backward compatibility.

To explain this limitation, we propose the concept of behavioral affordances to describe a crucial subset of functionality that HTML is currently incapable of expressing. We use behavioral affordances to assess the benefits and limitations of Triptych and suggest how these limitations might be filled by a standards-compatible semantic.

¹<https://alexanderpetros.com/triptych>

2 Background

A distributed hypermedia system such as the WWW consists of various components that allow it to function:

- A core hypermedia format (HTML)
- A core network protocol (HTTP)
- Hypermedia servers (web servers)
- Hypermedia clients (web browsers)

A review of each of these components will help us see what realistic modifications can be made to the web platform to support generalized hypermedia controls.

2.1 HTML

Pages on the WWW are created via the HyperText Markup Language (HTML), a hypertext derived from Standard Generalized Markup Language (SGML) [7]. The two most significant hypermedia controls in HTML are anchor tags, which implement the defining hyperlink mechanic [32] of the WWW, and forms, which transformed the WWW from a read-only hypermedia system into a read/write medium in which what today are called web applications can be built.

These two hypermedia controls are also the only significant user-interactive hypermedia controls available in HTML. Hyperlink anchors have been part of HTML since its inception, and the form element was added to the HTML 2.0 specification, which was published in 1995 [28]. Since then, no additional elements have been added to HTML that allow WWW users to obtain choices and select actions in terms of hypermedia interactions. Anchor and form elements comprised the core implementation of the uniform interface constraint outlined by Fielding in 2000, and they still do so today [9].

2.2 HTTP

The core network protocol of the WWW is the HyperText Transfer Protocol (HTTP), a stateless, application-level, text-based protocol for transferring HTML documents and other content [12, 17, 23]. HTTP requests can be driven by user selections of actions, via hypermedia controls, or by non-interactive hypermedia controls such as image elements.

The primary semantics of an HTTP request are its *target* and *method* [11]. Each request targets a resource on a server, identified by a Uniform Resource Locator (URL), and includes a method that describes the request’s intent with respect to that resource. For instance, a GET-method request simply asks the server to provide the client with the resource that can be found at that URL.

HTTP requests also provide a simple text-based name-value pair mechanism for transmitting metadata about an HTTP request. These are called *headers*, and they are included after the start-line of the request.

Listing 1 shows an HTTP request that would be issued to retrieve the `index.html` document from the server `www.example.com`.

Listing 1: An HTTP 1.1 GET request.

```
GET /index.html HTTP/1.1
Host: www.example.com
```

In addition to GET, the HTTP 1.1 specification defines a number of other methods. Of those methods, GET, POST, PUT, and DELETE are

frequently used by developers in web applications [2]. The PATCH method, added later via an extension proposal [8], is also sometimes used. In broad terms, GET is used for information retrieval, and POST, PUT, PATCH and DELETE are used for updating resources.²

In order to describe how the resource is to be modified, POST, PUT, and PATCH requests typically contain a *request body*. These bodies contain arbitrary data that the client provides along with the request. Although GET and DELETE are also capable of sending content in the request body, those semantics are left deliberately undefined, and doing so is generally discouraged. Instead, GET requests typically encode user-generated content as URL query parameters.

Listing 2 shows a basic POST request to the `/signup` relative URL. Note that the request includes form data—the user’s email—which is required to fulfill its purpose—signing that email for up for the service.

Listing 2: An HTTP 1.1 POST request.

```
POST /signup HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 24

email=erin%40example.com
```

HTTP methods can additionally be classified by two primary characteristics: whether or not the method is *safe*, and whether or not it is *idempotent*.

Methods that are safe should not produce any side-effects on the server. By denoting methods this way, server authors indicate that this resource may be fetched relatively freely—whether by user action, browser optimization, or automated web crawler. While servers may generate side-effects based on safe requests, clients that send safe requests are not to be held responsible for those side effects. The GET and HEAD methods are safe.

Methods are considered idempotent if the effect of $N > 0$ requests to a given URL has the same effect as a single request to that URL. PUT, which replaces the resource at the URL with the contents of the request body, is idempotent, because multiple PUT requests would just keep replacing the resource with an identical one; POST, which can be used to create a new resource, is not idempotent, because multiple POST requests might create multiple resources.

Table 1: HTTP methods and their properties.

Method	Safe?	Idempotent?	In HTML?
GET	Yes	Yes	Yes
POST	No	No	Yes
PUT	No	Yes	No
PATCH	No	No	No
DELETE	No	Yes	No

A surprising limitation of HTML is that, without additional enhancement via scripting, it is not possible to issue PUT, PATCH or DELETE requests. This is demonstrated in Table 1, which denotes the properties of a couple HTTP methods which are relevant to

²There is no requirement that a POST, in particular, update anything—the semantics of a POST are determined entirely by the server.

this paper. Note that in addition to the semantic limitations of the HTML-available HTTP methods—a POST describes a very different resource mutation than a DELETE—there are technical limitations as well: it is currently impossible for HTML authors to describe an unsafe, idempotent operation in hypermedia-based web applications.

2.3 Web Servers

In order to participate in the WWW, a server must implement the HTTP protocol, accepting HTTP requests and responding with HTTP responses. Hypermedia servers for the WWW are numerous [10, 20, 21, 34]; many programming languages have web server functionality directly integrated into the core library [14, 15, 27]. The large number of such servers indicates relative simplicity of such software, likely an important contributing factor to the growth of the WWW as a hypermedia system.

2.4 Web Browsers

Web browsers are the hypermedia clients for the WWW. As a hypermedia client, they implement the crucial functionality of interpreting and displaying hypermedia content for a user with a rich visual user experience, such as displaying clickable links, submit-able forms, etc.

With the deprecation and retirement of Internet Explorer, there are now three major browser engines: WebKit (Safari), Gecko (Firefox), and Chromium (Chrome) [4, 24].

3 Triptych: Towards General Hypermedia Controls

In order to extend HTML while preserving its present and future utility as a distributed hypermedia system, any proposal to add generalized hypermedia controls to HTML must therefore abide by two important constraints:

- New HTML features must be backwards compatible with existing clients, insofar as unrecognized tags and attributes will behave predictably and harmlessly on older clients.
- New HTML features must be forwards compatible with plausible future changes, with reasonable fallbacks that allow for today’s clients to be resilient to tomorrow’s servers.

Triptych is a set of three proposals that generalizes hypermedia controls in the manner described by [18]—and prototyped by htmx³—while abiding by the constraints required to actually deploy these enhancements to the WWW. They are, in order:

- (1) Support PUT, PATCH, and DELETE methods in HTML forms
- (2) Allow buttons to make arbitrary network requests, without being wrapped in forms
- (3) Support partial DOM replacement from HTTP responses

3.1 Additional HTTP Methods

The simplest, and most important, of the Triptych proposals is the addition of PUT, PATCH, and DELETE support to the HTML form element. Triptych’s approach is to extend the existing `method` attribute, which currently only supports GET and POST, with three new possible values: PUT, PATCH, and DELETE.

³<https://htmx.org>

Listing 3 shows a simple sign-up form, which uses the PUT method.

Listing 3: A simple sign-up form with PUT support.

```
1 <form action="/signup" method="PUT">
2   <input type="email" name="email">
3   <button>Submit</button>
4 </form>
```

Variations on this feature have been proposed many times [3, 19, 25]. Triptych’s proposal is mostly similar, but strictly limited in scope to just these three methods; this simplifies adoption of the change for browser vendors and increases the likelihood of the proposal being accepted.

Adding additional methods to HTTP forms is hampered by an unfortunate aspect of the HTML spec: if a browser does not recognize the value of the `method` attribute, it will default to a GET method. As discussed in Section 2.2, GET-method requests typically encode user-generated content as URL query parameters, while POST-method forms add them to the request body. HTML forms follow this pattern, so a form that makes an unintentional GET request exposes its contents to the URL, potentially leaking sensitive data.⁴

Even if support for additional HTTP methods were to be immediately available in browsers, this security concern would inhibit its adoption on the WWW. Web servers would need to ensure a web browser supports this HTML feature before utilizing it, otherwise the unrecognized method would result in GET requests being sent for PUT-method forms, possibly leaking sensitive data to the URL.

To resolve this, Triptych includes a fallback mechanism. An additional attribute, `custommethod`, can be specified to override the value of the `method` attribute, as seen in Listing 4.

Listing 4: A sign-up form in current HTML.

```
1 <form
2   action="/signup"
3   method="POST"
4   custommethod="PUT"
5 >
6   <input type="email" name="email">
7   <button>Submit</button>
8 </form>
```

This pattern introduces a novel form of *progressive enhancement* to HTML. Progressive enhancement typically refers to additional, non-essential functionality that is layered on top of the hypermedia with scripting; the website still functions properly without scripting, but the additional functionality enhances the experience when available. In this case, however, progressive enhancement occurs based on whether the hypermedia itself recognizes certain novel features.

Using the `custommethod` attribute, web applications can immediately start taking advantage of the idempotency semantics of PUT where available, but fall back to POST on clients where the `custommethod` attribute is not recognized. The server supports this

⁴Information encoded in the URL might leak through browser history, logs, or referrer headers.

by simply handling the `/signup` URL identically for both PUT and POST methods.

In this manner, Triptych is able to enhance HTML semantics while remaining infinitely backwards-compatible with old clients, fulfilling the mandate to “degrade gracefully” [29]. Once an application developer determines that PUT-method support is sufficiently widespread for their use-case, they can dispense with `custommethod` altogether.

The `custommethod` attribute is also forwards-compatible with full support for custom HTTP methods. The HTTP specification permits servers to define and use their own non-standard methods—functionality that has previously been proposed for HTML [19]. In order to limit scope, Triptych itself does not propose the ability to use non-standard HTTP methods in the form, but it is designed to make that feature easier to add in the future.

3.2 Button Actions

The second Triptych proposal expands the functionality of HTML buttons, enabling them to make network calls independent of a form element or script execution. It simply adds the `action` and `method` attributes from the form element to the button element.

Listing 5: A simple login form.

```

1 <form action="/session" method="POST">
2   <input type="email" name="email">
3   <input type="password" name="password">
4   <button>Login</button>
5 </form>
```

To illustrate the value of this feature, consider the hypermedia controls involved in basic web authentication, one of the most common patterns in web applications. Listing 5 shows how one might implement a traditional authentication form, with an email and password.

Note the HTTP semantics described by the form’s attributes: the user “logs in” to the website by creating (method=“POST”) a session resource (action=“/session”). As shown in Listing 6, with Triptych, HTML is able to simply and concisely model “logging out” as well.

Listing 6: A logout button.

```

1 <button action="/session" method="DELETE">
2   Logout
3 </button>
```

Unlike logging in, which relies on form data to verify the user’s identity, logging out can be described entirely using HTTP semantics: delete this session. No form body is necessary. By adding additional HTTP methods and making them available to the button directly, Triptych makes it possible to specify that interaction declaratively.

Listing 7: A logout button in a form wrapper.

```

1 <form action="/logout" method="POST">
2   <button>Logout</button>
3 </form>
```

Although this pattern is ubiquitous on the WWW, it cannot be accomplished with existing HTML controls. The closest analogue,

shown in Listing 7, requires the author to wrap the button in a superfluous form and issue a POST request to a related resource.

This pattern does work, but it obscures the intended hypermedia semantics in two important ways.

First, because the only unsafe method available to HTML is POST, the session must be deleted by referencing a different URL than the one used to create it. This eliminates the concept of a single resource from the URL entirely, which can have negative downstream effects on the application’s coherence. For instance, many server frameworks support file-based routing, in which the function that handles a particular URL route can be found at a corresponding file path in the server code. When sessions can be managed at the same URL, the authentication code can be co-located within the same file, making it easier to find, modify, and encapsulate that functionality.

Second, the parent form serves no purpose other than to allow its child button to make an HTTP request; the logout button straightforwardly does not have any form data. The unnecessarily verbose interface likely leads many developers to conclude that HTML is simply insufficient to describe even the most common interactive application functionality, and they often opt for scripted alternatives which do not introduce unnecessary semantic overhead.

Listing 8: A form with multiple actions.

```

1 <form action="/apply" method="POST">
2   <!-- inputs omitted for clarity -->
3
4   <button>Submit</button>
5   <button action="/" method="GET">
6     Cancel
7   </button>
8 </form>
```

Button actions also enable richer form controls. Consider a basic job application form as depicted in Listing 8. In this example, submitting the form with the unadorned “Submit” button will issue a POST request to the `/apply` route. Clicking the “Cancel” button, however, will ignore the parent form’s attributes and inputs; instead, it will make use of the button’s own attributes, which specify both a different URL and HTTP method. Unlike the logout flow, which can reasonably be achieved using a form wrapper, this pattern is extremely difficult to accomplish with standard HTML, because forms cannot contain other forms.

Once again, Triptych engages with the existing HTML terminology to make its new features as intuitive and unobtrusive as possible. It re-uses the `action` and `method` attributes from the form element, to preserve existing semantics. Servers can also progressively enhance their buttons by wrapping them in a form; clients that support the feature will see that the button has an `action` and ignore the parent form.

3.3 Response Placement

The final Triptych proposal dramatically expands the possibilities of what HTML can accomplish with the hypermedia response, by providing the ability to specify page replacement targets without performing full page navigations [6, 22].

Page navigation is an essential paradigm on the WWW, but it carries a lot of conceptual baggage. Almost every network request that can be specified by HTML results in a page navigation—both links and forms perform one by default. Page navigation creates a browser history entry, impacts back button behavior, and, semantically, asks the user to understand the new page as a new location.

Although navigation is the correct way to model the basic structure of a website, there are many hypermedia interactions that require network requests but should not alter the current resource location. While such interactions are already possible through scripting, Triptych makes it possible to model them declaratively. This is done by extending the `target` attribute of links and forms to accept arbitrary CSS selectors, along with a handful of additional keywords.

Listing 9: A "lock" button that replaces itself.

```

1 <button
2   action="/rooms/123/lock"
3   method="PUT"
4   target="_this">
5   Lock
6 </button>
```

Listing 9 shows a button that might be available in an online chatroom. This button locks Room #123, the room currently being viewed by the user, so that no newcomers are able to join. This action is modeled in HTTP as a PUT request to the `/rooms/123/lock` endpoint, and the button indicates that the contents of the HTTP response should replace it on the page, using the `_this` keyword as a parameter for `target`. This mirrors the style of existing `target` values, like `_blank` and `_parent` [33].

Listing 10: An "unlock" button that replaces itself.

```

1 <button
2   action="/rooms/123/lock"
3   method="DELETE"
4   target="_this">
5   Unlock
6 </button>
```

When the user clicks the “Lock”, the server responds with the “Lock” button’s inverse: an “Unlock” button for that same room. As shown in Listing 10, this is represented in HTTP as a DELETE request on the current lock.

Note that neither of these actions navigate the user to a new room. They both take place within the confines of the page `/rooms/123`, and consequently do not create new entries in the browser history. This is appropriate when the actions do not change which resource is being viewed, but instead modify the existing resource.

By returning an “Unlock” button, the server is applying Hypertext as the Engine of Application State (HATEOAS). The presence of the “Unlock” button confirms that, having completed the request to lock Room #123, the current state of that room resource is “Locked”, and that one of the actions the user can take to modify that resource is to “Unlock” it. A navigation to that page, later or by a different user, would simply include the button that corresponds to whichever action the user is allowed to take, given the room’s state

at the time of page load. We can say that the addition of response placement allows for *subnavigational* HATEOAS: instead of driving state updates with page navigation, subnavigational updates alter the current page to reflect the new state of the current resource.

It is very important that the server be aware of whether or not the request came from a client that is capable of placing responses. If the server only returns a small part of the page to a client that did not understand—and therefore ignored—the `target` attribute, the user might be navigated to an incomplete view. This can be resolved with a header that indicates the response’s target, and, in the absence of such a header, the server can safely assume that it is a full-page navigation. Once again, like with the `custommethod` attribute, Triptych provides the ability for the server to participate in backwards compatibility. The server can alter its response based on the client capabilities that the header—or lack thereof—implies, providing a smooth upgrade path via progressive enhancement.

4 Comparative Expressive Power

Triptych, with its goal of being adopted into web browsers, can only make relatively constrained extensions to HTML. To understand the limitations of this approach, we use the hypermedia control mechanism outlined in [18], which consist of four primitives: event triggers, request issuance, resource location, and response placement.

Of these four primitives, Triptych conclusively enhances two: the request issuance—by adding more HTTP methods—and the response placement—by generalizing the `target` attribute. It does not alter HTML’s ability to URLs, because no enhancement is necessary, nor does it impact the triggering event primitive. Whatever mechanism the client provided for selecting buttons and links before (e.g., a mouse click, a screen tap, etc.) remains unchanged.

In this section, we present a sampling of web user experience (UX) patterns that are possible with generalized hypermedia controls, sourced from `htmx`. We evaluate which patterns Triptych-enhanced HTML is capable of achieving, and which ones it is not. In doing so, we establish the basis for a formal categorization of those patterns, which is used in Section 5 to describe the semantic limitations that currently prevent HTML from reaching its potential as generalized hypermedia.

4.1 Newly Obtainable UX Patterns

The following UX patterns are a subset of common patterns not achievable in plain HTML. With the addition of Triptych, HTML is able to achieve some of them, meaningfully improving its expressive capabilities.

4.1.1 Click To Edit. The click to edit pattern allows a user to click a button in order to trigger an editing view—usually to edit the resource at the current URL. In HTML, this can only be modeled as a page navigation, where viewing the data and editing the data take place on separate pages. Using partial page replacement, Triptych makes it possible to switch between these modes on the same page.

Listing 11: An editable form.

```

1 <div id="target">
2   Text To Edit
3   <button method="get">
```

```

4 |         action="/edit"
5 |         target="#target">
6 |     Click To Edit
7 | </button>
8 | </div>

```

In Listing 11, a Triptych-enhanced button issues a request to the `/edit` relative URL and replaces the entire contents of the element with the ID `target` (i.e. the `div` enclosing the button.) A similar pattern for saving the form is enabled by Triptych.

4.1.2 Click To Load. Websites often break up data into smaller segments, loading the first chunk quickly and then giving the user the opportunity to load more at a later time. Traditionally, a website might accomplish this by paginating, where each page provides another chunk of data. Modern applications, however, often append the new data to the existing page.

One way to add this pattern to a webpage is with a “Click to Load” button, which allows the user to load the additional elements on command. Listing 12 shows an HTML implementation of the pattern, taking advantage of Triptych’s response placement functionality.

Listing 12: A button that loads more resources.

```

1 | <li>Item N + 1</li>
2 | <!-- Additional items omitted for clarity-->
3 | <li id="target">
4 |     Item 2N
5 |     <button
6 |         method="get"
7 |         action="/page/3"
8 |         target="#target">
9 |         Load More...
10 |     </button>
11 | </li>

```

In this case, the button targets the enclosing `li`, and will replace it with the next page of items, retrieved from the sever. The server response should also include the current `2N` list item, to preserve its place in the overall list.

4.2 Unobtainable UX Patterns

While Triptych, in its current form, does extend the expressive power of HTML, there remain many UX patterns found in modern web applications that are not achievable.

4.2.1 Infinite Scroll. Infinite scroll is a UX pattern in which, when a particular element in a series (e.g. a list) is scrolled into view, new elements are appended to the series. It is essentially an enhanced click-to-load, removing the requirement for explicit user action entirely. Listing 13 is an implementation of infinite scroll using `htmx` markup.

Listing 13: An `htmx` implementation of infinite scroll.

```

1 | <li>Item N + 1</li>
2 | <!-- Additional items omitted for clarity-->
3 | <li hx-trigger="revealed"
4 |     hx-get="/page/3"
5 |     hx-target="closest ul"

```

```

6 |     hx-swap="beforeend">
7 |     Item 2N
8 | </li>

```

Achieving a similar UX pattern with a pure hypermedia implementation in Triptych is not possible. None of Triptych’s extensions add the ability for a hypermedia exchange to be triggered based on the user’s scroll position. By contrast, `htmx` is able to achieve this through the `hx-trigger=revealed` attribute, which changes the triggering event from a user click to a specific user scroll position,

4.2.2 Active Search. Active search is a UX pattern where, as a user types into a text input, the search results are updated immediately. In `htmx`, this behavior can be achieved with `hx-trigger`, shown in Listing 14.

Listing 14: An `htmx` implementation of active search.

```

1 | <input type="search"
2 |       name="search"
3 |       hx-get="/search"
4 |       hx-trigger="keyup delay:500ms"
5 |       hx-target="#search-results">
6 | <div id="search-results">
7 | </div>

```

Once again, Triptych is not able to implement this pattern, since the triggering event is not a click, but the release of a keypress (i.e., `keyup`). Triptych’s partial DOM replacement does make it possible for the target of the search to be a partial page replacement—but triggering that search still requires a click.

4.3 Comparative Analysis

Although the patterns presented in this section are only a small subset of the user experiences that can be achieved through generalized hypermedia controls, they highlight which of the control primitives HTML is currently best primed to support.

Triptych succeeds in bringing support for three of the four hypermedia control primitives to HTML. HTML currently has very limited support for customizing the request issuance and the response placement. However, Triptych is able to easily enhance those primitives, since HTML already contains semantic controls to customize them. Triptych simply makes those existing controls—the method and target attributes—more capable.

The remaining primitive demonstrates Triptych’s most glaring limitation. Generalized hypermedia libraries like `htmx` can initiate a hypermedia exchange from diverse triggers such as scroll position, element lifecycle, and timers. HTML, by contrast, is largely limited to user selections via clicks, which open links, or submissions of forms. Triptych does not attempt to generalize this primitive, because there are no readily available event trigger controls to extend in HTML.

We conclude that while Triptych represents a significant advancement in the expressiveness of HTML, it falls short of the expressive power of fully generalized hypermedia controls due to its inability to further customize the event triggers. This is a necessary limitation for Triptych, which adopts a strategy of minimal extensions to well-established HTML semantics. HTML itself has

no established notion of modifying event triggers, and therefore nothing for Triptych to extend.

5 Behavioral Affordances

In James Gibson’s “Theory of Affordances,” he describes the “affordances” of an environment as what it offers, provides, or furnishes the animal [16]. Affordances are equally a fact of the environment and a fact of the agent that is interacting with the environment.

In a hypermedia context, affordances are typically discussed in terms of the mechanisms by which the hypermedia user and the hypermedia client interact with one another to achieve the goals of the hypermedia user.

We define *behavioral affordances* as the subset of affordances that a hypermedia provides that respond to behavioral—though not necessarily interactive—stimulus from the user [26, 31]. When we say that HTML lacks the ability to customize the event trigger, we mean that, more abstractly, it lacks explicit and general behavioral affordances. The HTML environment provides buttons and links that the user can interact with using a click, tap, or analogous action—and little else.

While HTML is limited in this regard, web browsers do provide a rich event model. Authors can use scripting to capture those events and trigger hypermedia exchanges based on a wide variety of stimuli. What is currently missing is a mechanism to translate those events, which the browser already emits, into declarative affordances.

5.1 Existing Behavioral Affordances

While HTML does not provide an explicit and general behavioral affordance mechanism to authors and users, there are specialized behavioral affordance mechanisms.

5.1.1 Lazy Loading. The first example of a specialized behavioral affordance is the `loading` attribute [13], which can be used on `img` and `iframe` tags. Both images and iframes are interesting hypermedia controls in that unlike, for example, links, they dispatch their hypermedia request independent of direct user interaction. Instead of triggering when the user clicks an element on the page, both of these controls issue a request immediately when a page is loaded, issuing an HTTP GET request for the resource at the URL specified by the `src` attribute.

While an HTML author does not have direct control over this behavior, they can modify it to an extent by setting the `loading` attribute on an `img` or `iframe` to `lazy`. In this case, the web browser will defer loading the given image or `iframe` until it comes within a calculated distance of the viewport (the distance is browser specific.) This is essentially the same affordance that enables the infinite scroll pattern from 4.2.1, but it is limited to images and iframes, and does not support general hypermedia constructs. It also gives the HTML author relatively little control over the triggering event, with the intersection calculation being opaque and browser-defined.

5.1.2 Datalists. Another specialized behavioral affordance can be in the HTML `datalist` element, which can be associated with an input element to provide autocomplete behavior. A simple usage of the `datalist` element, and its `option` element children, can be seen in Listing 15.

Listing 15: A simple usage of HTML datalists.

```

1 | <label>Color
2 |   <input name="color" list="colors">
3 | </label>
4 | <datalist id="colors">
5 |   <option value="red"></option>
6 |   <option value="green"></option>
7 |   <option value="blue"></option>
8 | </datalist>

```

In this example, when a user begins typing a letter, an auto-completion affordance will appear offering completions that match what the user has typed. For example, if a user types the character ‘r’ into the input, the value “red” will appear in a suggestions popup, allowing the user to select the desired value.

The behavioral affordance here is tied to the input or keyup event—depending on implementation—and, once again, is very limited in what it provides to HTML authors. While the `datalist` is specified entirely in hypermedia (i.e. the values of the autocomplete affordance are determined entirely by information found in the hypermedia itself), it does not allow an HTML author to specify a hypermedia exchange to take place.

This limits the effectiveness of this affordance to situations when the domain of autocomplete values is relatively small and known at the time the HTML document is served.

5.1.3 Collapsible Details. A final example of a limited behavioral affordance available in HTML is the `details` element. As with the `datalist`, this is not a hypermedia exchange active affordance, but it is a behavioral affordance. The `detail` element, when combined with a `summary` element, can be used to show only the summary of a section of text in an HTML document, and then reveal the additional content when the users clicks on the `summary` element.

This is typically rendered with a visual signifier such as a triangle indicating to users that the `summary` element is clickable.

Listing 16: A simple usage of HTML details.

```

1 | <details>
2 |   <summary>Hypermedia</summary>
3 |   A media that supports hypermedia controls
4 | </details>

```

Listing 16 will, when rendered in a web browser, show the text “Hypermedia” to the user, along with attendant visual affordance indicating that it is interactive. When a user clicks the text, the definition of hypermedia will be shown. As with the `datalist` attribute, no hypermedia exchange takes place in this case and the interaction is entirely local to the current HTML document.

5.2 User-Intentional vs Author-Intentional Affordances

A final important distinction between the various behavioral affordances we have discussed is between user-intentional affordances and author-intentional affordances.

A click event that triggers navigation occurs in response to active user input. The hypermedia-driven action therefore occurs in response to positive intentionality from the user.

The behavioral affordance offered by the lazy attribute, in contrast, is more nuanced. The triggering event occurs in response to user input scrolling the element into view—although this is a direct user action, a lower level of user intentionality involved. Indeed, the user may even be annoyed to see an image only begin to load once it is in the viewport. Here, the intentionality lies more in the HTML document author. They have chosen to use the lazy temporal modifier to save bandwidth, to make the initial page load faster for the user; the user did not take a conscious action with this result in mind.

It is important to note that the intentionality of the user and the HTML author do not always align. A mechanism for generalizing behavioral affordances in HTML may be abused by HTML authors to the detriment of web users.

5.3 Surfacing Behavioral Affordances

In our analysis of Triptych we showed that the primary constraint on its ability to implement the patterns found in more general hypermedia control systems was the lack of a natural mechanism to express behavioral affordances in HTML. While HTML currently has some behavioral affordances, they are specified either implicitly, as in the case of anchor tags, or in an ad-hoc manner, as in the case of the lazy attribute. Surfacing behavioral affordances in a markup-friendly manner would enable HTML authors and users to enjoy a richer hypermedia environment on the Web.

In htmx, behavioral affordances are available via the `hx-trigger` attribute, which allows the author to specify events—and modifiers of those events—that trigger a hypermedia exchange. HTML could take this approach by adding a `trigger` attribute to request-capable elements. For many common cases, this approach is simple and flexible, but it also has limitations. The `hx-trigger` implementation has a relatively complex syntax for modifiers that does not align with the syntax found on existing HTML attributes.

Hyperview, a mobile hypermedia presented alongside htmx in [18], takes an element based approach, specifying behavioral affordances via a behavior element.

Listing 17: A hyperview implementation of generic behavioral affordances.

```

1 <view style="Button">
2   <behavior trigger="press"
3     href="/display" />
4   <behavior trigger="longPress"
5     href="/edit"
6     target="new" />
7   <text style="Button_Label">
8     Item
9   </text>
10 </view>
```

The button in Listing 17 will issue a request to `/display` after a normal-length button press to display the element. If the user long-presses the button, however, it issue a request to `/edit` and display an editing UI for that item.

The Hyperview approach uses the natural nesting of SGML-based markup language elements to express behaviors and it is a promising direction to explore for surfacing behavioral affordances

in HTML. Multiple behavioral affordances can be naturally specified via multiple behavior elements, nested within a single parent. Modifiers to behaviors can be added via attributes in a manner that is easy for clients to parse and interpret,

This hypothetical behavior element could be composed with Triptych to implement patterns such as the infinite scroll, as seen in Listing 18. In doing so, it completes the generalization of HTML’s hypermedia controls that Triptych alone is incapable of.

Listing 18: A possible HTML implementation for generic behavioral affordances.

```

1 <li>Item N + 1</li>
2 <!-- Additional items omitted for clarity-->
3 <li id="item_2n">
4   <form action="/page/3"
5     method="get"
6     target="#item_2n">
7     <behavior trigger="revealed"/>
8   </form>
9   Item 2N
10 </li>
```

6 Conclusion

In this paper, we conducted an analysis of Triptych, a set of three proposals to extend HTML’s existing attributes with richer functionality. We assessed the expressive power afforded to HTML authors by those proposals, comparing them against ideal implementations with fewer constraints, and noting HTML’s limited ability to customize the triggering event in particular. We introduced the concept of behavioral affordances to explain why Triptych is unable to fully implement generalized hypermedia controls in HTML, and presented a hypothetical mechanism for customizing the triggering event in a manner that conforms to HTML’s existing syntax and patterns.

This research identifies a significant need—and opportunity—for HTML to add semantics that capture many of the dynamic user interactions that have arisen on the web in the past decades. Such a mandate prescribes work of a kind that has not been done on the HTML standard since the introduction of the form element, 30 years ago. As HTML has no natural mechanism for customizing triggering events today, adding behavioral affordances to HTML will require careful research, design, and consultation.

The rewards for doing so are considerable. A declarative mechanism for customizing the triggering event would dramatically increase the ease of deploying sophisticated interactions to the web, increasing its value to both developers and users. Standardizing this behavior in HTML affords more control to humans users, by allowing them to consume the content in a form factor that best suits their needs, and to machine users, which can interpret and interact with the hypertext APIs more precisely. A web where authors specify behaviors in explicit markup elements, rather than JavaScript, is a web that is more open, intuitive, and accessible than the one we have today.

References

- [1] WHATWG. [n.d.]. Is there a process for removing bad ideas from a standard? <https://whatwg.org/faq#removing-bad-ideas> [Online; accessed 08-March-2025].
- [2] Alexander Petros and Carson Gross. 2024. Support PUT, PATCH, and DELETE in HTML Forms. <https://alexanderpetros.com/triptych/form-http-methods> [Online; accessed 08-May-2025].
- [3] Mike Amundsen. 2011. Support PUT and DELETE with HTML FORMS. <http://amundsen.com/examples/put-delete-forms/> [Online; accessed 17-March-2025].
- [4] Michael Amundsen. 2017. *RESTful web clients : enabling reuse through Hypermedia* (first edition. ed.). O'Reilly, Beijing, [China].
- [5] Kenneth M. Anderson, Mark Bernstein, Kasper Østerbye, and Leslie Carr. 1997. Integrating open hypermedia systems with the World Wide Web. In *Hypertext '97 : Southhampton, UK, April 6-11 1997 : Eighth ACM Conference on Hypertext : proceedings*. ACM, New York, NY, USA, 157–166.
- [6] Mark Anderson, Leslie Carr, and David E Millard. 2017. There and here: patterns of content transclusion in wikipedia. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media*. ACM, Prague, Czech Republic, 115–124.
- [7] James Clark. 1997. Comparison of SGML and XML. *World Wide Web Consortium Note* 15 (1997).
- [8] Lisa M. Dusseault and James M. Snell. 2010. PATCH Method for HTTP. RFC 5789. doi:10.17487/RFC5789
- [9] Roy Thomas Fielding. 2000. REST: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California* (2000).
- [10] Roy T. Fielding and Gail Kaiser. 2002. The Apache HTTP server project. *IEEE Internet Computing* 1, 4 (2002), 88–90.
- [11] Roy T. Fielding, Mark Nottingham, and Julian Reschke. 2022. HTTP Semantics. RFC 9110. doi:10.17487/RFC9110
- [12] Roy T. Fielding, Mark Nottingham, and Julian Reschke. 2022. HTTP/1.1. RFC 9112. doi:10.17487/RFC9112
- [13] Mozilla Foundation. 2025. The loading attribute. <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img#loading>
- [14] OpenJS Foundation. 2025. Node.js Documentation. <https://nodejs.org/api/http.html> [Online; accessed 23-April-2025].
- [15] Python Software Foundation. 2025. The Python Standard Library. <https://docs.python.org/3/library/http.server.html> [Online; accessed 23-April-2025].
- [16] James J. (James Jerome) Gibson. 1979. *The ecological approach to visual perception*. Houghton Mifflin, Boston.
- [17] David. Gourley and Brian. Totty. 2002. *HTTP : the definitive guide* (first edition. ed.). O'Reilly, Sebastopol, California.
- [18] Carson Gross, Dillon Shaffer, and Matt Revelle. 2024. Hypermedia Controls: Feral to Formal. In *Proceedings of the 35th ACM Conference on Hypertext and Social Media (Poznan, Poland) (HT '24)*. Association for Computing Machinery, New York, NY, USA, 52–64. doi:10.1145/3648188.3675127
- [19] W3C Working Group. 2015. W3C HTML Form HTTP Extensions. <https://www.w3.org/TR/form-http-extensions> [Online; accessed 17-March-2025].
- [20] Eric Dean Katz, Michelle Butler, and Robert McGrath. 1994. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN systems* 27, 2 (1994), 155–164.
- [21] Clément Nedelcu. 2013. *Nginx HTTP server*. Packt Publishing, Birmingham.
- [22] Theodor Holm Nelson. 1995. The heart of connection: hypermedia unified by transclusion. *Commun. ACM* 38, 8 (1995), 31–33.
- [23] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. 1999. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. doi:10.17487/RFC2616
- [24] Pavel Panchekha and Chris Harrelson. 2025. *Web Browser Engineering*. Oxford University Press. doi:10.1093/9780198913887.001.0001 arXiv:<https://academic.oup.com/book/59224/book-pdf/61200133/isbn-9780198913887.pdf>
- [25] Julian Reschke. 2010. Consider adding support for PUT and DELETE as form methods. https://www.w3.org/Bugs/Public/show_bug.cgi?id=10671 [Online; accessed 17-March-2025].
- [26] Alexander Stoytchev. 2005. Behavior-grounded representation of tool affordances. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, IEEE, New York, 3060–3065.
- [27] Go Core Team. 2025. Go HTTP library. <https://pkg.go.dev/net/http> [Online; accessed 20-March-2025].
- [28] W3C. [n.d.]. Form Elements. https://www.w3.org/MarkUp/html-spec/html-spec_toc.html#SEC8.1 [Online; accessed 15-February-2024].
- [29] W3C. 2025. HTML Design Principles. <https://www.w3.org/TR/html-design-principles> [Online; accessed 23-April-2025].
- [30] Jill Walker. 2005. Feral hypertext: when hypertext literature escapes control. In *Proceedings of the Sixteenth ACM Conference on Hypertext and Hypermedia (Salzburg, Austria) (HYPERTEXT '05)*. Association for Computing Machinery, New York, NY, USA, 46–53. doi:10.1145/1083356.1083366
- [31] Huifen Wang, Jialu Wang, and Qiuhong Tang. 2018. A review of application of affordance theory in information systems. *Journal of Service Science and Management* 11, 1 (2018), 56–70.
- [32] Noah Wardrip-Fruin. 2004. What hypertext is. In *Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*. ACM, New York, NY, USA, 126–127.
- [33] WHATWG. [n.d.]. HTML Living Standard. <https://html.spec.whatwg.org/multipage/> [Online; accessed 15-February-2024].
- [34] Wikipedia. 2025. Comparison of web server software. https://en.wikipedia.org/wiki/Comparison_of_web_server_software [Online; accessed 20-March-2025].