Preserving REST-ful Visibility Of Rich Web Applications With Generalized Hypermedia Controls

CARSON GROSS AND MATTHEW REVELLE
HyperMedia Research Group @ Montana State University

This research investigates how the visibility benefits of the REST architectural style can be preserved through the use of generalized hypermedia controls without sacrificing user experience in web applications. JavaScript, an implementation of the code-on-demand constraint of REST, has been used to enable rich user experiences on the World Wide Web (WWW) but its use has increasingly become detrimental to visibility in modern web applications. We review the benefits provided by visibility to the WWW and the tension between visibility and code-on-demand. The concept of generalized hypermedia controls is presented and we demonstrate how generalized hypermedia controls preserve visibility while enabling a rich user experience in real-world web applications.

1. INTRODUCTION

A benefit of the REST [Fielding and Taylor 2000] network architecture of the World Wide Web (WWW) is *visibility*, in which the networking semantics of components are visible [Jacobs and Walsh 2004] to intermediaries [Barrett and Maglio 1998], allowing these intermediaries to add value to the overall system with features such as caching [Davison 2001] [Erman et al. 2009] and preloading [Yeo et al. 2020]. As web-based applications have become increasingly sophisticated the visibility of these applications has diminished, with their network behavior being specified in opaque *code-on-demand* [Carzaniga et al. 2007], that is JavaScript-based logic. Generalized hypermedia controls [Gross et al. 2024] offer a mechanism for providing the advanced user experiences expected of modern web applications while retaining the visibility benefits inherent to the REST architectural style.

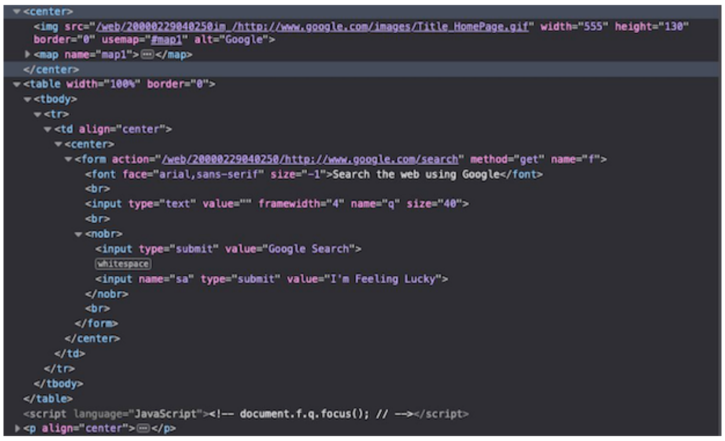
In his thesis defining the REST network architecture, Fielding states:

> [Architectural styles] can also influence the visibility of interactions within a network-based application by restricting interfaces via generality or providing access to monitoring. Visibility in this case refers to the ability of a component to monitor or mediate the interaction between two other components.

One of the primary mechanisms by which REST improves visibility is requiring the use of hypermedia. Hypermedia encodes network interactions directly in responses, allowing intermediate components in a REST-ful system to ”see” these interactions and potentially act on or modify them.

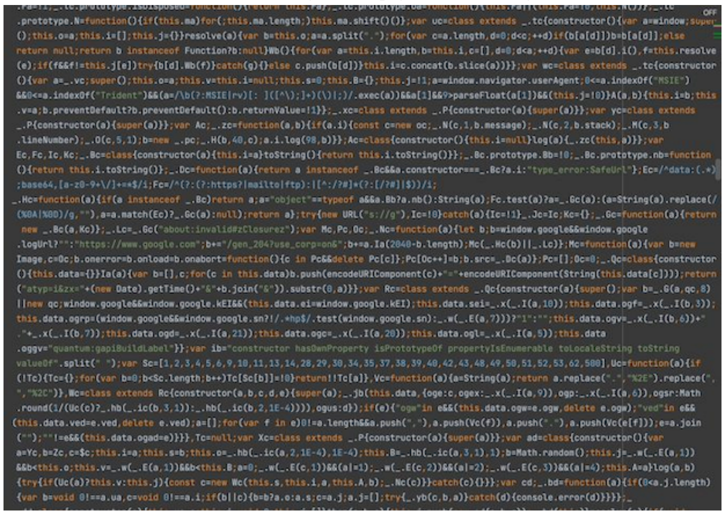
This advantage is not only restricted to intermediate software components. Human users are also able to take advantage of this visibility via the ”View Source” [Doctrow 2024] feature of web browsers, which allows users to inspect the hypermedia source of web pages and to understand how they operate.

This user-facing visibility is a key component of the openness [Davis et al. 1993] and success of the WWW as a distributed system: it has allowed users to learn from implementations they encounter as they interact with web pages.



```html
▾<center>
   <img src="/web/20000229040250im_/http://www.google.com/images/Title_HomePage.gif" width="555" height="130"
   border="0" usemap="#map1" alt="Google">
  ▸<map name="map1">⋯</map>
 </center>
▾<table width="100%" border="0">
  ▾<tbody>
    ▾<tr>
      ▾<td align="center">
        ▾<center>
          ▾<form action="/web/20000229040250/http://www.google.com/search" method="get" name="f">
             <font face="arial,sans-serif" size="-1">Search the web using Google</font>
             <br>
             <input type="text" value="" framewidth="4" name="q" size="40">
             <br>
            ▾<nobr>
               <input type="submit" value="Google Search">
               whitespace
               <input name="sa" type="submit" value="I'm Feeling Lucky">
             </nobr>
             <br>
           </form>
         </center>
       </td>
     </tr>
   </tbody>
 </table>
 <script language="JavaScript"><!-- document.f.q.focus(); // --></script>
▸<p align="center">⋯</p>
```

(a) Source code from the Google home page in 2000. [Google LLC 2000].



```javascript
.prototype.N=function(){if(this.ma)for(;this.ma.length;)this.ma.shift()()};var uc=class extends _.tc{constructor(){var a=window;super
();this.o=a;this.i=[];this.j={}}resolve(a){var b=this.o;a=a.split(".");for(var c=a.length,d=0;d<c;++d)if(b[a[d]])b=b[a[d]];else
return null;return b instanceof Function?b:null}Wb(){for(var a=this.i.length,b=this.i,c=[],d=0;d<a;++d){var e=b[d].i(),f=this.resolve
(e);if(f&&f!=this.j[e])try{b[d].Wb(f)}catch(g){}else c.push(b[d])}this.i=c.concat(b.slice(a))}};var wc=class extends _.tc{constructor
(){var a=_.vc;super();this.o=a;this.v=this.i=null;this.s=0;this.B={};this.j=!1;a=window.navigator.userAgent;0<=a.indexOf("MSIE")
&&0<=a.indexOf("Trident")&&(a=/\b(?:MSIE|rv)[: ]([^\);]+)(\)|;)/.exec(a))&&a[1]&&9>parseFloat(a[1])&&(this.j=!0)}A(a,b){this.i=b;this
.v=a;b.preventDefault?b.preventDefault():b.returnValue=!1}};_.xc=class extends _.P{constructor(a){super(a)}};var yc=class extends
_.P{constructor(a){super(a)}};var Ac;_.zc=function(a,b){if(a.i){const c=new oc;_.N(c,1,b.message);_.N(c,2,b.stack);_.M(c,3,b
.lineNumber);_.O(c,5,1);b=new _.pc;_.H(b,40,c);a.i.log(98,b)}};Ac=class{constructor(){this.i=null}log(a){_.zc(this,a)}};var
Ec,Fc,Ic,Kc;_.Bc=class{constructor(a){this.i=a}toString(){return this.i.toString()}};_.Bc.prototype.Bb=!0;_.Bc.prototype.nb=function
(){return this.i.toString()};_.Dc=function(a){return a instanceof _.Bc&&a.constructor===_.Bc?a.i:"type_error:SafeUrl"};Ec=/^data:(.*)
;base64,[a-z0-9+\/]+=*$/i;Fc=/^(?:(?:https?|mailto|ftp):|[^:/?#]*(?:[/?#]|$))/i;
_.Hc=function(a){if(a instanceof _.Bc)return a;a="object"==typeof a&&a.Bb?a.nb():String(a);Fc.test(a)?a=_.Gc(a):(a=String(a).replace(/
(%0A|%0D)/g,""),a=a.match(Ec)?_.Gc(a):null);return a};try{new URL("s://g"),Ic=!0}catch(a){Ic=!1}_.Jc=Ic;Kc={};_.Gc=function(a){return
 new _.Bc(a,Kc)};_.Lc=_.Gc("about:invalid#zClosurez");var Mc,Pc,Oc;_.Nc=function(a){let b;b=window.google&&window.google
.logUrl?"":"https://www.google.com";b+="/gen_204?use_corp=on&";b+=a.Ia(2040-b.length);Mc(_.Hc(b)||_.Lc)};Mc=function(a){var b=new
Image,c=Oc;b.onerror=b.onload=b.onabort=function(){c in Pc&&delete Pc[c]};Pc[Oc++]=b;b.src=_.Dc(a)};Pc=[];Oc=0;_.Qc=class{constructor
(){this.data={}}Ia(a){var b=[],c;for(c in this.data)b.push(encodeURIComponent(c)+"="+encodeURIComponent(String(this.data[c])));return
("atyp=i&zx="+(new Date).getTime()+"&"+b.join("&")).substr(0,a)}};var Rc=class extends _.Qc{constructor(a){super();var b=_.G(a,qc,8)
||new qc;window.google&&window.google.kEI&&(this.data.ei=window.google.kEI);this.data.sei=_.x(_.I(a,10));this.data.ogf=_.x(_.I(b,3));
this.data.ogrp=(window.google&&window.google.sn?!/.*hp$/.test(window.google.sn):_.w(_.E(a,7)))?"1":"";this.data.ogv=_.x(_.I(b,6))+"
."+_.x(_.I(b,7));this.data.ogd=_.x(_.I(a,21));this.data.ogc=_.x(_.I(a,20));this.data.ogl=_.x(_.I(a,5));this.data
.oggv="quantum:gapiBuildLabel"}};var ib="constructor hasOwnProperty isPrototypeOf propertyIsEnumerable toLocaleString toString
valueOf".split(" ");var Sc=[1,2,3,4,5,6,9,10,11,13,14,28,29,30,34,35,37,38,39,40,42,43,48,49,50,51,52,53,62,500],Uc=function(a){if
(!Tc){Tc={};for(var b=0;b<Sc.length;b++)Tc[Sc[b]]=!0}return!!Tc[a]},Vc=function(a){a=String(a);return a.replace(".","%2E").replace(",
","%2C")},Wc=class extends Rc{constructor(a,b,c,d,e){super(a);_.jb(this.data,{oge:c,ogex:_.x(_.I(a,9)),ogp:_.x(_.I(a,6)),ogsr:Math
.round(1/(Uc(c)?_.hb(_.ic(b,3,1)):_.hb(_.ic(b,2,1E-4)))),ogus:d});if(e){"ogw"in e&&(this.data.ogw=e.ogw,delete e.ogw);"ved"in e&&
(this.data.ved=e.ved,delete e.ved);a=[];for(var f in e)0!=a.length&&a.push(","),a.push(Vc(f)),a.push("."),a.push(Vc(e[f]));e=a.join
("");""!=e&&(this.data.ogad=e)}}},Tc=null;var Xc=class extends _.P{constructor(a){super(a)}};var ad=class{constructor(){var
a=Yc,b=Zc,c=$c;this.i=a;this.s=b;this.o=_.hb(_.ic(a,2,1E-4),1E-4);this.B=_.hb(_.ic(a,3,1),1);b=Math.random();this.j=_.w(_.E(a,1))
&&b<this.o;this.v=_.w(_.E(a,1))&&b<this.B;a=0;_.w(_.E(c,1))&&(a|=1);_.w(_.E(c,2))&&(a|=2);_.w(_.E(c,3))&&(a|=4);this.A=a}log(a,b)
{try{if(Uc(a)?this.v:this.j){const c=new Wc(this.s,this.i,a,this.A,b);_.Nc(c)}}catch(c){}}};var cd;_.bd=function(a){if(0<a.j.length)
{var b=void 0!==a.ua,c=void 0!==a.i;if(b||c){b=b?a.o:a.s;c=a.j;a.j=[];try{_.yb(c,b,a)}catch(d){console.error(d)}}}};_
```

(b) Source code from the Google home page in 2023. [Google LLC 2023].

1.1 The Decline of Visibility On The Web

There has been a marked decline in visibility in web applications since Fielding's thesis. This is a trend that has accelerated with the rise of what are known today as *Single Page Application* (SPA) web applications.

Single Page Applications use a single web page to bootstrap JavaScript-based—rather than hypermedia-based—web applications. These web applications typically communicate with servers via a plain data format such as JavaScript Object Notation (JSON)[1] rather than via a hypermedia format, thus abandoning the REST architectural style [Feng et al. 2009].

The reduction in the visibility that has occurred due to this shift can be seen in dramatic fashion on one of the most popular websites in the world: the Google search engine. The source of the Google home page in the year 2000 is shown in Figure 1a, retrieved from the Wayback Machine[2]. Note in particular that the *network actions* that the page offers are clearly visible in the source hypermedia, in particular via the `action` attribute found on the `form` element, which specifies what network endpoint the form will submit its request to.

Contrast this visibility with a small portion of the Google home page source in the year 2023, shown in Figure 1b. Google has moved from what was a hypermedia-based approach for their home page to a JavaScript-based approach. They now communicate with their remote servers using data formats rather than hypermedia.

This change was done for good reasons: the user experience of the Google search page in 2023 is more advanced than the 2000 version. However, it is apparent this advance in user experience has come at the cost of visibility.

2. BACKGROUND

The REST architectural style consists of a series of constraints that should be adhered to for a particular distributed system to be considered REST-ful. The two most critical constraints of REST with respect to visibility are: the *uniform interface* constraint and the optional *code-on-demand* constraint.

2.1 The Uniform Interface

The uniform interface constraint is of particular importance in the REST architectural style. Fielding [Fielding and Taylor 2000] states:

> The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components.

[1]Confusingly, JSON-based APIs are often referred to as "REST" APIs today in both industry and academic settings. This confusion of language has lead to significant misunderstandings around the REST network architecture.

[2]https://web.archive.org

This uniform interface constraint is further divided into four additional sub-constraints. First, the system *must* use a mechanism for the identification of resources (e.g. URLs). Second, the system *must* perform manipulation of resources through representations. Third, the system *must* use self-descriptive messages; and finally, the system *must* use hypermedia as the engine of application state (HATEOAS).

Within the uniform interface, these last two sub-constraints—the use of self-describing messages and hypermedia—are the primary mechanism by which REST achieves its distinctive visibility. The network interactions are embedded in hypermedia in the form of *hypermedia controls*—elements that encode or imply network interactions with remote systems.

2.2 Code-On-Demand

The code-on-demand constraint of REST is considered optional: it allows, but does not require, a REST-ful system to support the downloading and execution of code in a client. This simplifies REST-ful systems by allowing new features to be implemented after system deployment.

At the time of Fielding's earlier works the code-on-demand constraint on the WWW was satisfied by technologies such as Java Applets and JavaScript. Today, JavaScript is the dominant code-on-demand technology [Worldmetrics.org 2024] on the WWW.

Fielding noted in [Fielding and Taylor 2000] that this optional code-on-demand constraint, in contrast with the uniform interface, has the potential to harm the visibility benefits of REST-ful systems:

> [Code-on-demand] simplifies clients by reducing the number of features required to be pre-implemented. Allowing features to be downloaded after deployment improves system extensibility. However, it also reduces visibility, and thus is only an optional constraint within REST.

As is apparent in the contrast between the Google home pages from 2000 and 2023, Fielding was correct in his prediction that the utilization of code-on-demand would indeed reduce visibility in REST-ful systems.

3. GENERALIZED HYPERMEDIA CONTROLS

In our recent paper [Gross et al. 2024] we presented a generalization of the concept of hypermedia controls. Hypermedia controls are elements found embedded in a hypermedia document that specify network interactions with remote servers and allow users to select non-linear actions in that document.

The canonical example of a hypermedia control is the hyperlink which, in the standard implementation, allows a user to click on a text element and ”jump” to another place in the current document or to another document entirely.

In our paper we review four hypermedia controls found in HTML [Walker 2005]: anchor elements, form elements, image elements and IFrame elements. We note that in all four cases we have the following pattern: first, an event triggers an HTTP request of some method type. The event is user-initiated in the case of anchors and forms, and browser-initiated in the case of images and IFrames. Next, the hypermedia response to that request either replaces the current document or is inserted into the existing document in some manner. We noted that, while anchors (links) and forms often replace the entire document, images and IFrames instead use a *transclusional* model, where content is placed within the existing document rather than replacing it. We also noted that anchors and forms are capable of this behavior via the *target* attribute, in conjunction with IFrame elements.

From these observations we derived a functional definition of hypermedia controls:

> A hypermedia control is an element that responds to *an event* trigger by *issuing a type of request* to a URL and placing the response *at some position* within the user agent's viewport.

This functional definition then points to a generalization of the concept of hypermedia controls within a hypermedia format: allow *any* event to initiate *any* sort of request on *any* element and place the response content *anywhere* in the document (i.e., generalized transclusion [Nelson 1995]).

A hypermedia that implements this generalization will increase the expressiveness of the REST system it participates in, allowing for more interactive patterns to be adopted within that network architecture.

3.1 An Implementation of Generalized Hypermedia Controls for the Web

We presented multiple implementations of this generalization of hypermedia controls in real world hypermedia systems in [Gross et al. 2024]. One of the implementations presented is htmx, which bring this generalization to HTML. htmx allows HTML authors to define new hypermedia controls in HTML via a set of declarative attributes that closely follow the existing hypermedia-active attributes.

By increasing the expressiveness of HTML as a hypermedia format within the declarative framework already established for existing hypermedia controls, htmx makes it possible to create significantly richer user experiences within the original hypermedia-oriented and REST-ful network architecture of the web.

UX patterns that can be achieved with htmx include: *infinite scroll*, loading more content when a user reaches the bottom of a page; *lazy loading* of content, allowing a request to be made for additional content only after the initial page has been loaded; and *inline editing*, where a user may edit the details directly within a UI context without requiring a full page refresh.

3.2 Google Instant Search and Hypermedia Controls

Google introduced ”Instant Search” [Mayer 2010] in 2010. This feature allowed search results to be displayed to users as they typed, rather than requiring them to submit a form to perform the search as had previously been needed. This dynamic user interface was not achievable with any existing HTML infrastructure and thus required the use of JavaScript to implement. It remains the primary feature of the current Google home page that differentiates it's functionally from its earlier version in the 2000s.

While the increased use of JavaScript on the Google home page has harmed the visibility of that page in REST terms it has enabled an improved user experience. However, it is worth considering if the improved user experience can be implemented in a manner that retains the visibility benefits of a REST-ful, hypermedia-based system.

We are glad to demonstrate that the answer is ”yes”; the additional expressive power of generalized hypermedia controls can achieve a very similar user experience without losing visibility.

This can be accomplished with an htmx-powered input element that has been, through attributes, transformed into a hypermedia control as shown here:

```html
<input type="search"
       name="search"
       hx-get="/search"
       hx-trigger="keyup delay:500ms"
       hx-target="#search-results"
       hx-swap="innerHTML">
<div id="search-results">
</div>
```

The `ht-get` attribute tells htmx that this `input` element should issue an HTTP `GET` request to `/search` when a triggering event occurs. This attribute converts the `input` element from an inert input into a hypermedia control. Note the visibility of the network interaction present here.

The `ht-trigger` attribute specifies that the triggering event is a `keyup` event. This is followed by a `delay` modifier that prevents requests from being issued until there is a 500 millisecond pause in the stream of `keyup` events. This use of a delay is known as *debouncing* the event and prevents the control from overloading the remote server with requests as the user types.

The `hx-target` attribute specifies the target element into which the response content will be placed with a CSS selector. In this case the selector, `#search-results`, specifies the `div` tag found immediately following the `input` element.

Finally, the `ht-swap` attribute indicates that the response content should replace the inner HTML of the target element. This could alternatively specify that the response content replace the element entirely, be appended to the element, etc. These last two attributes implement generalized transclusion in HTML.

It is important to recognize that the response content in this case is expected to be in a hypermedia format. htmx does not replace the standard web model of network exchanges via hypermedia in favor of a data format like JSON, but rather utilizes the existing hypermedia infrastructure of the WWW and the browser.

A web developer is able to create an interactive experience that mirrors the Instant Search functionality introduced by Google with these four htmx attributes. The results of the search will be displayed in the `div` element below the `input` element as a user types.

4. CONCLUSION

In this paper, we identified and described the impact of code-on-demand on visibility in the REST network architecture. We then presented generalized hypermedia controls as an alternative technology for achieving modern user interfaces on the web.

We demonstrated that, with htmx, we are able to implement dynamic user interfaces while retaining the visibility benefits of the REST network architecture. Network interactions remain visible in the hypermedia, in particular in the `hx-get` attribute, providing both intermediate components and end users the ability to inspect and modify these interactions.

The visibility benefits of the REST network architecture of the WWW do not need to be abandoned due to the interactive deficiencies of HTML. By increasing the expressiveness of HTML with generalized hypermedia controls we can retain these benefits while providing the interactive experiences users have come to expect on the modern web.

ACKNOWLEDGMENTS

Many thanks to Mike Amundsen for reviewing this article and suggesting improvements.

REFERENCES

BARRETT, R. AND MAGLIO, P. P. 1998. Intermediaries: New places for producing and manipulating web content. *Computer Networks and ISDN Systems 30,* 1-7, 509–518.

CARZANIGA, A., PICCO, G. P., AND VIGNA, G. 2007. Is code still moving around? looking back at a decade of code mobility. In *29th International Conference on Software Engineering (ICSE'07 Companion)*. IEEE, Los Alamitos, CA, 9–20.

DAVIS, H., HALL, W., HEATH, I., HILL, G., AND WILKINS, R. 1993. Towards an integrated information environment with open hypermedia systems. In *Proceedings of the ACM Conference on Hypertext*. ECHT '92. Association for Computing Machinery, New York, NY, USA, 181–190.

DAVISON, B. D. 2001. A web caching primer. *IEEE Internet Computing 5,* 4, 38–45.

DOCTROW, C. 2024. Hypercard presaged the web's critical "#ViewSource" affordance. `https://x.com/doctorow/status/1701934612686196872`. Accessed: 2024-10-26.

ERMAN, J., GERBER, A., HAJIAGHAYI, M. T., PEI, D., AND SPATSCHECK, O. 2009. Network-aware forward caching. In *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. Association for Computing Machinery, New York, NY, USA, 291–300.

FENG, X., SHEN, J., AND FAN, Y. 2009. REST: An alternative to RPC for web services architecture. In *2009 First International Conference on Future Information Networks*. IEEE, Institute of Electrical and Electronics Engineers, New York, NY, USA, 7–10.

FIELDING, R. T. AND TAYLOR, R. N. 2000. Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine. AAI9980887.

GOOGLE LLC. 2000. Google homepage source code (2000). `https://web.archive.org/web/20000229040250/http://www.google.com/`. Accessed: 2024-10-26.

GOOGLE LLC. 2023. Google homepage source code (2023). `https://web.archive.org/web/20231211000742/https://www.google.com/`. Accessed: 2024-10-26.

GROSS, C., SHAFFER, D., AND REVELLE, M. 2024. Hypermedia controls: Feral to formal. In *Proceedings of the 35th ACM Conference on Hypertext and Social Media*. HT '24. Association for Computing Machinery, New York, NY, USA, 52–64.

JACOBS, I. AND WALSH, N. 2004. Architecture of the world wide web, volume one - W3C recommendation. Tech. rep., The World Wide Web Consortium. Accessed: 2024-10-27.

MAYER, M. 2010. Search: now faster than the speed of type. `https://googleblog.blogspot.com/2010/09/search-now-faster-than-speed-of-type.html`. Accessed: 2024-10-30.

NELSON, T. H. 1995. The heart of connection: hypermedia unified by transclusion. *Communications of the ACM 38,* 8, 31–33.

WALKER, J. 2005. Feral hypertext: When hypertext literature escapes control. In *Proceedings of the Sixteenth ACM Conference on Hypertext and Hypermedia*. HYPERTEXT '05. Association for Computing Machinery, New York, NY, USA, 46–53.

WORLDMETRICS.ORG. 2024. JavaScript statistics reveal dominance in websites, job market, and more. `https://worldmetrics.org/javascript-statistics/`. Accessed: 2024-10-26.

YEO, J., RIM, J.-H., SHIN, C., AND MOON, S.-M. 2020. Accelerating web start-up with resource preloading. In *Web Engineering*, M. Bielikova, T. Mikkonen, and C. Pautasso, Eds. Springer International Publishing, Cham, 37–52.

Carson Gross is a lecturer and PhD student at Montana State University and is the founder of the HyperMedia Research Group. Their latest publication is entitled "Hypermedia Controls: Feral to Formal" and has been published in Proceedings of the 35th ACM Conference on Hypertext and Social Media.

Matthew Revelle is an assistant professor of Computer Science at Montana State University. His research interests include the use of formal methods and machine learning for discovering and reasoning over software vulnerabilities and the design of software systems.