
# IIPC Future of the Web Workshop – Introduction & Overview

**May 17, 2012**

# TABLE OF CONTENTS

**IIPC Future of the Web Workshop Goals** ..... 3

**Classical Web Crawlers/Web Archiving** ..... 4

**Current Challenges** ..... 6

- **Database driven features and functions** ..... 6
- **Complex/variable URI formats and inconsistent/variable link implementations** ..... 7
- **Dynamically generated, ever changing, URIs** ..... 8
- **Rich Media** ..... 9
- **Scripted, incremental display and page loading mechanisms** ..... 10
- **Scripted, HTML forms** ..... 11
- **Multi-sourced, embedded material** ..... 12
- **Dynamic login/auth services: captchas, cross-site/social authentication, & user-sensitive embeds** ..... 13
- **Alternate display based on user agent or other parameters.** ..... 14
- **Exclusions by convention** ..... 14
- **Exclusions by design** ..... 14
- **Server side scripts & remote procedure calls** ..... 15
- **HTML5 "web sockets"** ..... 16
- **Mobile publishing** ..... 18

**Current Mitigation Strategies** ..... 20

- **Configure a classical crawler for capture** ..... 20
- **Extend a classical crawler** ..... 20
- **Deposit an instance of a "site" in a virtual instance with an archive** ..... 21

## IIPC 'Future of the Web' Workshop Goals

The Future Web workshop was held, Thursday, May 3, 2012 in Washington DC as part of the 2012 International Internet Preservation Consortium General Assembly meeting (IIPC GA), hosted by the Library of Congress. There were approximately sixty to seventy attendees at the workshop. Notes from the session are available in a separate document.

The primary audience for this workshop was the engineer who is attempting to address the challenges of the "current and Future" Web in the implementation of web archiving programs for their own institutions or for others. We wanted to keep the level of dialogue as technical as possible and we largely succeeded in meeting that goal. We wanted to exit the workshop with some specific engineering investigations and research problems that we believe need funding by IIPC and others to move the community forward. Those ideas and recommendations are included at the end of this document.

After an introductory session, we held three 40-minute panels:

- Capture, looking at the problems of obtaining the content to be preserved.
- Redisplay/Replay, looking at the problems of recreating the user experience from the preserved content.
- Scale, looking at the problems of doing both of these at Web scale.

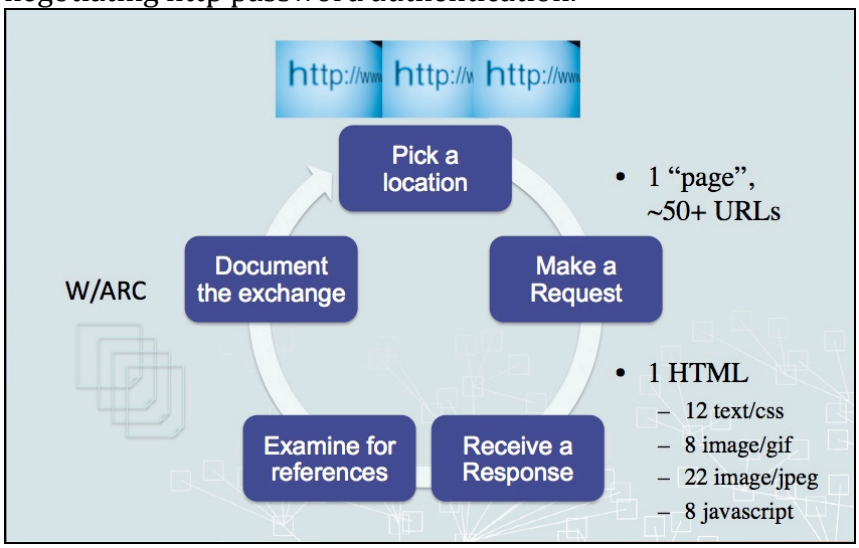
Each panel had 3 or 4 panelists, who each had ~5 minutes to share their thoughts on the topic. Then we opened discussions to Q&A. Notes from the opening remarks, panel sessions, and Q&A have been summarized in a separate document.

To follow is the introductory material used to provide a shared context for all participants (panelists and audience members). At the conclusion of the workshop we edited this introduction to include the issues raised by panelists and attendees not originally covered by this text, and added commentary about which mitigation approaches are really feasible in an operational vs. a theoretical context.

# INTRODUCTION

## Classical Web Crawlers/Web Archiving

The original or classical model of archiving the web was built on hypertext – http and html standards (http://www.w3.org/2003/glossary/alpha/H/) - that primarily involved the tasks of parsing text, identifying and following links and occasionally negotiating http password authentication.




**Source: original diagram created by G. Mohr in 2007**

In the 1990's and even moving into the early 2000's, most Web pages and site designs were relatively simple. Many used "hybrid" site models that enabled publishers to separate out resources into separate directories and to optimize for different usage scenarios and workflows, but the majority of the emphasis was on more traditional load balancing for scale vs integration across diverse hosts and services. The interaction with the site visitor was fairly straightforward, although even then, some features and functions were supported by the browser and some were enabled by the web/app server such as:

- search services
- database driven forms
- remote procedure calls used to execute tasks and functions, and to "personalize" page display, etc.

***Example: A Hybrid-Style Web Site circa 1998***

*The hybrid style of Web site organization allowed you to put some common files (such as often-used graphics) in separate directories and to organize unique files with their related HTML pages. In this example, there is an HTML document called products.html, located at the URL* ***http://www.example.com/products/products.html****. Some of the graphics are maintained in a subdirectory of the main directory of this Web site; the subdirectory is called graphics/. There are also links to other pages in the main directory and in a subdirectory called about/.*

***Listing 10.2 basetag.html Creating a Directory Structure***

```
<HTML>
<HEAD>
<TITLE>Our Products</TITLE>
<BASE HREF="http://www.example.com/">
</HEAD>
<BODY>
<IMG SRC="products/prod_ban.gif">
<H2>Our Products</H2>
<P>Here's a listing of the various product types we have available. Click the name of the product
category for more information:</P>
<DL>
<DT>
<DD><IMG SRC="graphics/bullet.gif"> <A HREF="products/pc_soft.html">
PC Software</A>
<DD><IMG SRC="graphics/bullet.gif"> <A HREF="products/mac_soft.html">
Macintosh Software</A>
<DD><IMG SRC="graphics/bullet.gif"> <A HREF="products/pc_hard.html">
PC Hardware</A>
<DD><IMG SRC="graphics/bullet.gif"> <A HREF="products/mac_soft.html">
Macinotsh Hardware</A>
</DL>
<HR>
<A HREF="index.html">Return to Main</A>
</BODY>
</HTML>
```

Even in the early days of the Web it was understood that anything NOT downloaded to the client, i.e. generated server side, and content excluded by convention (e.g. via robots.txt) or via design (e.g. sites architected to minimize or avoid capture) would impact the scope of the archiving and/or site indexing effort.

The classical model of web archiving was helped along when the Web was largely a graph of page-nodes that anyone could observe (both the pages and the links between them). Even some of its then-barriers -- robots, logins, forms -- just hid parts of that graph; overcoming the barriers (with permission, or alternative URL-discovery mechanisms and site maps) could extend the usefulness model, revealing more sub-graphs of content that still largely fit the model. Despite the problems, it was possible using relatively simple techniques and little customization to collect a useful representation of the current state of the Web in a reasonable timeframe at an affordable cost. And it was possible to replay the collected content in a way that, despite some imperfections, a wide range of users found valuable.

Over the last decade, at first gradually, and now more rapidly, the classical model is becoming obsolete. By 2000 greater emphasis was placed on integrating services maintained by distributed hosts/content aggregators, and by 2003 the dominance of social networks and user-generated content began to overtake more traditional publishing methodologies. At first this shift did not impact the “document” model of web publishing as site owners maintained two versions of their sites: one that was

crawler friendly to ensure premium placement in search results, and one that was "user differentiated" and often protected behind a login by a "walled garden". What has emerged in the past five years is an era of the "web site" as an interactive mash-up – a distributed network of data services and applications vs. a directory of documents. Site maps and other methods of serving "alternative versions" of a resource to a crawler persist but alternative versions place greater and greater emphasis on the text alone (as it is needed to index for live web search), rendering both the look and feel and the functionality offered by the alternative version less and less similar to the version presented to a user browsing to the site/service. These alternatives can work for optimizing the discovery of live web resources, but they do not provide archival quality replicas in support of cultural preservation and research.

For this and many other reasons (today, not just in the future!), the classical model of web archiving is no longer sufficient for

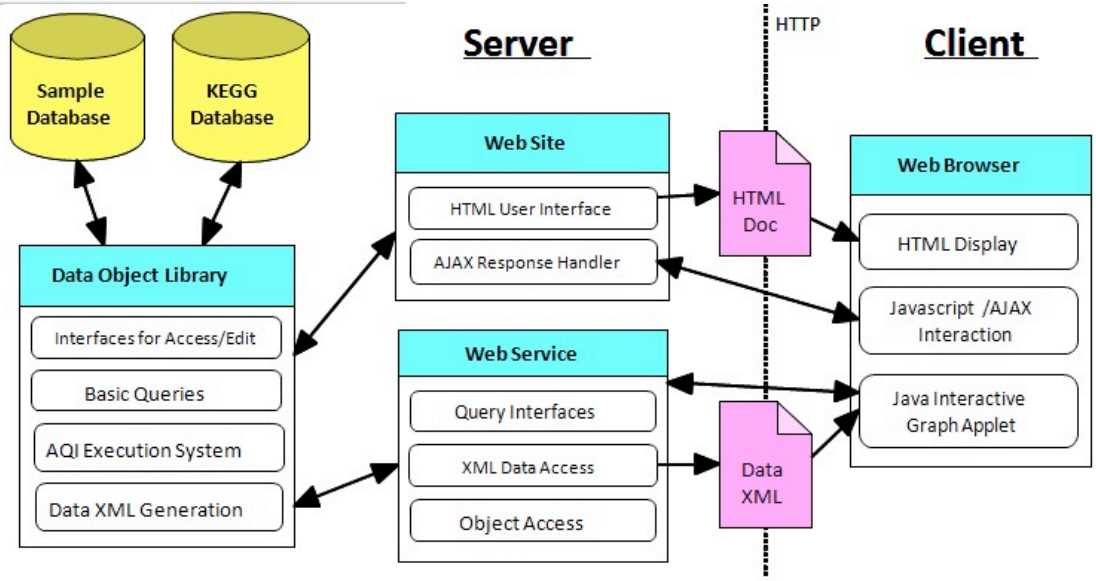
- capturing preserving, and re-rendering all the bytes of interest we care about
- representing the look and feel of a resource at the time it was archived
- recreating the behaviors an end user interacting with the site would have experienced at the time the resource was archived.

The list below highlights some of the current challenges presented by evolving publishing and consumption models and the changing requirements for archiving and preservation of "born digital" resources. This write-up will only begin to scratch the surface of each of these in the hopes of bounding each problem space, and possibly offering up examples of activities and other real-world efforts designed to try and circumvent and/or mitigate each. This list is compiled in no particular order. In fact, the ordering has zero significance – there is no relation between order and relative impact or importance of each challenge to the web archiving community. This list is simply a dump if ideas - a starting point from which we can foster live discussion and possibly future collaborations.

## Current Challenges

### *Database driven features and functions*

Database driven features and functions can be problematic for a classical crawler, especially if the database data is ever changing. Slow-changing databases are easier to think of as providing a set of documents that can be sampled; a fast-changing database outruns crawling and can add to some of the other factors detailed in this list. Certainly, a fast changing database will have profound impact on available crawler resources and might even start to mimic behaviors of spam traps in which crawler resources are tied up for indefinite periods of time trying to execute capture instructions for a never ending supply of new "data". The image below excerpted from […] serves to illustrate one approach to this type of architecture.




**Source: http://nashua.case.edu/PathwaysWeb/Images/PathCase.Architecture.png**

### *Complex/variable URI formats and inconsistent/variable link implementations*

Another key challenge to the classical crawl model is the prevalence of complex/variable URI formats and inconsistent/variable link implementations, especially URIs that support volatile services -- not quite 'documents' -- but that also still contribute to page-rendering. Changes to display that vary significantly by #hash portion are now a norm with some classes of sites, whereas in the past a #hash found in a URI largely meant that the #hash was just a pointer to a portion of a "document". Today, the URI w/ and w/o hash tag does not produce the same content.

A great write-up on how Twitter URIs functioned in December 2011, excerpted from the W3C Blog, is provided below as illustration of this phenomenon.

> **Now let's look at Twitter. What happens if I link to the tweet** http://twitter.com/#!/JeniT/status/35634274132561921? Although it's not indicated in the Vary header, Twitter determines what to do about any requests to this hashless URI based on whether I'm logged in or not (based on a cookie).
>
> If I am logged on, I get the new home page. This home page GETs (through various iframes and Javascript obfuscation) several small JSON files through Twitter's API: http://api.twitter.com/1/statuses/show.json?include_entities=true&contributor_details=true&id=35634274132561921: the details of the tweet
>
> http://api.twitter.com/1/statuses/35634274132561921/retweeted_by.json?count=15: details about retweets
>
> http://api.twitter.com/1/users/lookup.json?user_id=&screen_name=unhosted: details about the twitter user @unhosted, who was mentioned in the tweet.

This JSON gets converted into HTML and embedded within the page using Javascript. All the links within the page are to hash-bang URIs and there is no way of identifying the hashless URI (unless you know the very simple pattern that you can simply remove it to get a static page).

If I'm not logged on but am using a browser that understands Javascript, the browser GETs http://twitter.com/; the script in the returned page picks out the fragment identifier and redirects (using Javascript) to http://twitter.com/JeniT/status/35634274132561921.

If, on the other hand, I'm using curl or a browser without Javascript activated, I just get the home page and have no idea that the original hash-bang URI was supposed to give me anything different.

The response to the hashless URI http://twitter.com/JeniT/status/35634274132561921 also varies based on whether I'm logged in or not. If I am, the response is a 302 Found to the hash-bang URI http://twitter.com/#!/JeniT/status/35634274132561921. If I'm not, for example using curl, Twitter just returns a normal HTML page that contains information about the tweet that I've just requested.

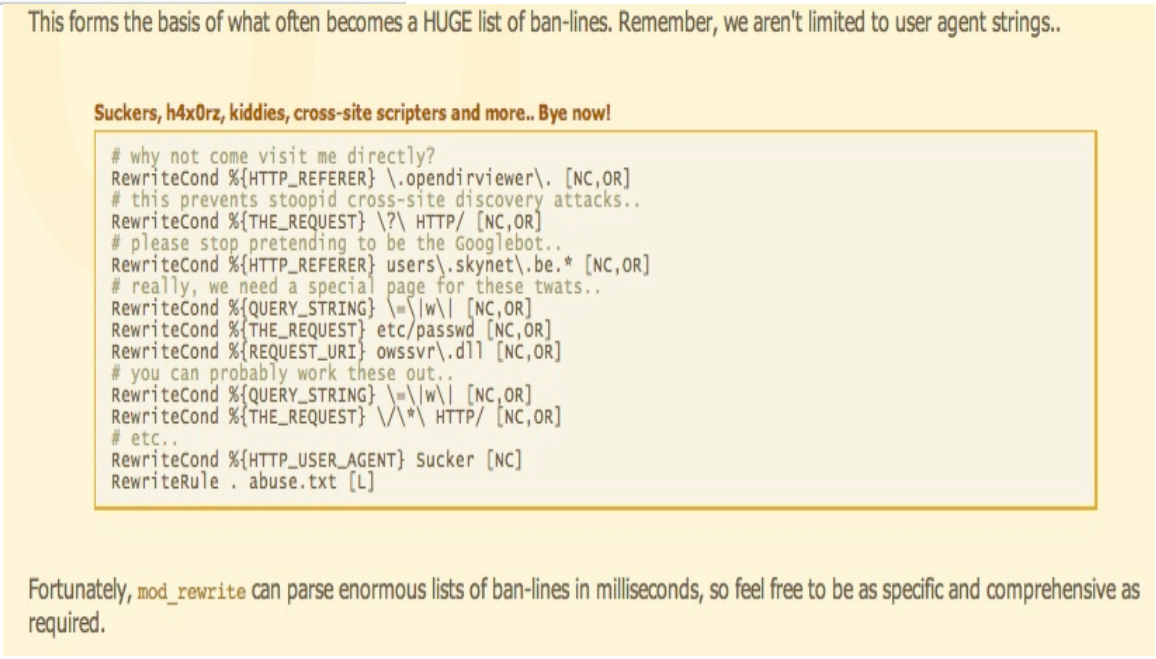
Finally, if I request the _escaped_fragment_ version of the hash-bang URI http://twitter.com/?_escaped_fragment_=/JeniT/status/35634274132561921 the result is a 301 Moved Permanently redirection to the hashless URI http://twitter.com/JeniT/status/35634274132561921 which can be retrieved as above.

Requesting a status that doesn't exist such as http://twitter.com/#!/JeniT/status/1 in the browser results in a page that at least tells you the content doesn't exist. Requesting the equivalent _escaped_fragment_ URI redirects to the hashless URI http://twitter.com/JeniT/status/1. Requesting this results in a 404 Not Found result as you would expect.

***Source:*** **W3C Blog: Hash URI's http://www.w3.org/QA/2011/05/hash_uris.html**

## *Dynamically generated, ever changing, URIs*

Dynamically generated, ever changing, URIs for serving the same resources and/or for credentialing visitors is not a new challenge for classical crawlers. In fact, volatile URIs have always been an issue. Some sites improved/altered site implementations for SEO purposes, but those that are providing web apps and are *not* SEO focused are backsliding on this, using URIs in unpredictable custom ways. (e.g. Apache .htaccess – "…mod_rewrite enables you to send browsers from anywhere to anywhere. You can create rules based not simply on the requested URL, but also on such things as IP address, browser agent (send old browsers to different pages, for instance), and even the time of day; the possibilities are practically limitless…." – An example appears below from: http://corz.org/serv/tricks/htaccess2.php, http://codesamplez.com/web-server/apache-htaccess-basics




Another example of dynamic URIs from: http://esi-examples.akamai.com/

```
<esi:comment
text="The following example dynamically builds the esi:include URL based on the current date.
An example source URL: /news/2001/08/16/index.html"/>

<esi:include src="/news/$strftime($time(), '%Y/%m/%d')/index.html" alt="/news/index.html"/>
```

When the dynamic URI contains the coordinates of the mousethis problem is even more pronounced. Capture might be possible but what can be done about replay?

### *Rich Media*

Rich media, even in downloadable form, has always been a challenge for classical crawlers as site owners often employ techniques to discourage bulk, distributed downloads via bots and other automated software applications. Streamed media, especially when it's combined with custom app interfaces or anti-collection measures, cannot be captured via a classical crawler. Alternate capture and replay methods must be employed.

The top five ad supported video sites (YouTube.com, Vimeo.com, Metacafe.com, Hulu.com, Veoh.com), each offer streaming video services (some also provide downloadable videos). Netflix and Amazon also offer streamed video access services as part of monthly or annual subscription fees.

Sample code for embedding a streaming media player and resource/s excerpted from: http://www.mediacollege.com/video/streaming/embed/

```
<OBJECT ID="MediaPlayer" WIDTH="192" HEIGHT="190" CLASSID="CLSID:22D6F312-B0F6-11D0-
94AB-0080C74C7E95"
STANDBY="Loading Windows Media Player components..." TYPE="application/x-oleobject">
<PARAM NAME="FileName" VALUE="videofilename.wmv">
<PARAM name="ShowControls" VALUE="true">
<PARAM name="ShowStatusBar" value="false">
<PARAM name="ShowDisplay" VALUE="false">
<PARAM name="autostart" VALUE="false">
<EMBED TYPE="application/x-mplayer2" SRC="videofilename.wmv"
NAME="MediaPlayer" WIDTH="192" HEIGHT="190" ShowControls="1" ShowStatusBar="0"
ShowDisplay="0" autostart="0"> </EMBED> </OBJECT>
```

### *Scripted, incremental display and page loading mechanisms*

Some of the most challenging web design choices involve scripted, incremental display and page loading mechanisms that do not require downloading of some or all resources used to create/enable the visual display and/or functions of a resource to the "client" visiting that resource. These approaches paired with the #hash-anchor problem detailed above, or as an 'optimization' or visual flourish, or in pursuit of 'app-like' look/feel can render classical methods of capture and re-rendering useless, or too resource intensive to configure and maintain for ongoing successful capture. Below is a code sample of an incremental page display implementations, excerpted from: http://www.codeproject.com/Articles/21106/Incremental-Page-Display-Pattern-for-User-Controls#

```
public class IncrementalLoader : System.Web.UI.WebControls.CompositeControl, IScriptControl
{
  protected override void OnInit(EventArgs e)
  {
    base.OnInit(e);
    // JavaScript registration - only happens for the first server control loaded
    if (!this.Page.ClientScript.IsClientScriptIncludeRegistered(
                  "__IncrementalLoaderScript"))
    {
      string resource = this.Page.ClientScript.GetWebResourceUrl(this.GetType(),
        "ControlLoading.IncrementalLoader.js");

      this.Page.ClientScript.RegisterClientScriptInclude(this.GetType(),
        "__IncrementalLoaderScript", resource);
    }
  }
  // Gets a path for the javascript to query.  This points to the HTTP handler
  public string GetCallbackPath()
  {
    string path = this.Page.Request.Path;
    if (!string.IsNullOrEmpty(path))
    {
      path = path.Replace(".aspx", HandlerExtension);
    }
    string controlLocation = this.Page.ResolveUrl(this.ResourceToLoad);
```

```
    string query = Utility.EncryptString(controlLocation);
    path += "?" + ControlQuery + "=" + HttpUtility.UrlEncode(query);
    return path;
  }
  // Each server control instance renders a single line of JavaScript to load
  // the user control when the browser is done loading the page.
  public string RenderLoadScript()
  {
    return "LoadContent('" + GetCallbackPath() + "', '" +
          _contentPanel.ClientID + "', '"
          + _loadingPanel.ClientID + "', '" +
          _faultPanel.ClientID + "');";
  }
  // Registers the startup script
  private void RetisterClientScript()
  {
    StringBuilder script =
      new StringBuilder("<script type="\"text/javascript\"">\r\n");
    this.Page.ClientScript.RegisterStartupScript(this.GetType(), _
                    contentPanel.ClientID +
                    "LoadingScript", script.ToString());
  }
          }
```

### *Scripted, HTML forms*

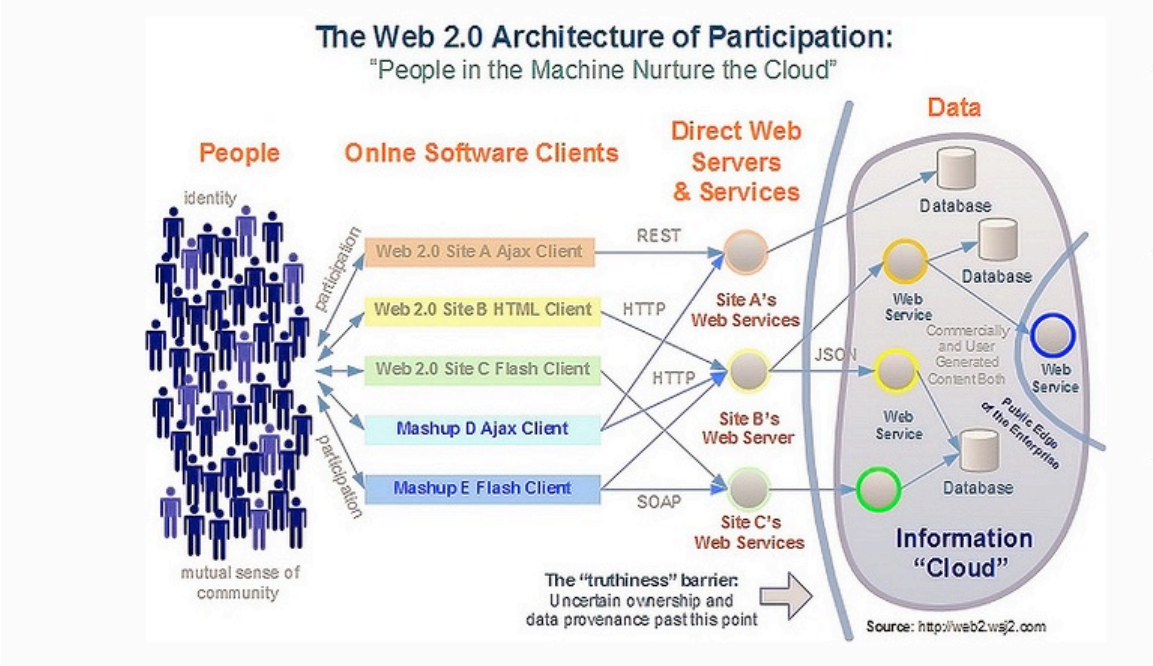
Form based entry that filters content and services made available to the visitor: aggravates the traditional 'deep web' problem. These combined with incremental/in-place reloads (so that all the possible 'looks' of a page don't have their own entry URLs) can inhibit collection by a classical crawler. These implementations most often manifest themselves as form based registration and/or login processes, but can be applied to other tasks as well. A leading example of this type of form-based, server side, site implementation is the Realtor.com website (www.realtor.com). Below is another example of how an html form can be converted to a complex, javascript object.

**Demo of form as complex javascript object (Src: http://jsfiddle.net/ct4Lb/ ):**

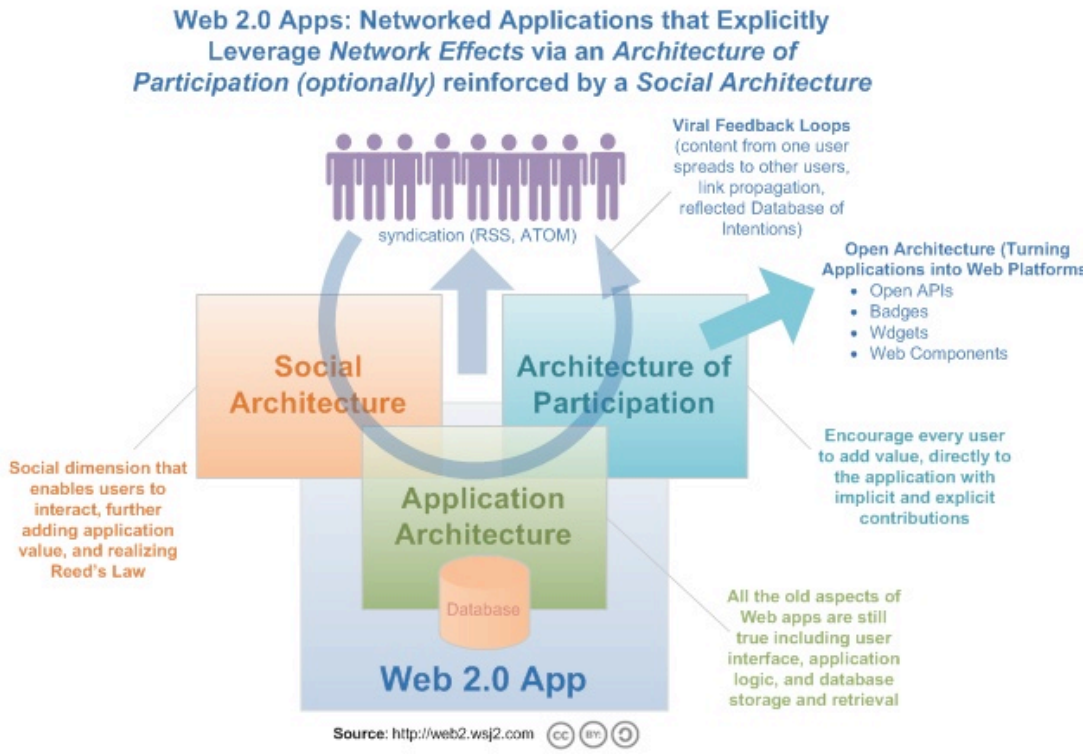
```
<form>
  <input name="Foo" value="1" />
  <input name="Parent.Child1" value="1" />
  <input name="Parent.Child2" value="2" />
</form>

$(function(){
   var form = document.forms[0];
   alert(form.Foo);
   alert(form['Parent.Child1'].value);
});
```

### *Multi-sourced, embedded material*

Multi-sourced, embedded material -- pulled from many sites, many non-document 'services', aggravates the age-old problem of getting everything to render in a "single page" in a tight time window. Some recent studies point to the average "Web page" being compiled of more than 65 files, more than half of which are served from separate hosts distributed around the globe. This issue is only exacerbated when the "user" is a conventional crawler attempting to guess at when the resources that are handed over to the client have finished loading. The images below illustrate just how many layers deep the embeds can extend.




**Source:** http://static.flickr.com/39/123558605_8446bd5471.jpg




**Source:** http://www.thewavingcat.com/wp-content/uploads/2007/05/web2appmodel.png

How do we identify provenance reliably in the context of layered, distributed service architectures and mash-ups?

### *Dynamic login/auth services: captchas, cross-site/social authentication, & user-sensitive embeds*

Dynamic login/auth services: captchas, cross-site/social authentication, also user-sensitive embeds ("your friends also liked..." on a page that's otherwise classically collectable) that require sequencing and maintenance of state. Below is a code example of a capcha embedded in a registration workflow excerpted from: http://www.html-form-guide.com/contact-form/html-contact-form-captcha.html

The HTML form below contains the fields for name, email and message. In addition, we have the CAPTCHA image. The <img> tag for the CAPTCHA image points to the script captcha_code_file.php. The PHP script in 'captcha_code_file.php' creates the image for the captcha and saves the code in a session variable named '6_letters_code'.

```
<form method="POST" name="contact_form"
action="<?php echo htmlentities($_SERVER['PHP_SELF']); ?>">
<label for="name">Name: </label>
<input type="text" name="name"
value="<?php echo htmlentities($name) ?>">
<label for="email">Email: </label>
<input type="text" name="email"
```

```
value="<?php echo htmlentities($visitor_email) ?>">
<label for="message">Message:</label>
<textarea name="message" rows=8 cols=30>
<?php echo htmlentities($user_message) ?></textarea>
<img src="captcha_code_file.php?rand=<?php echo rand(); ?>"
id="captchaimg" >
<label for="message">Enter the code above here :</label>
<input id="6_letters_code" name="6_letters_code" type="text">
<input type="submit" value="Submit" name="submit"></form>
```

### *Alternate display, experiences based on user agent (browser, mobile, etc.) - display alternatives on other parameters.*

Alternate display, experiences based on user agent (browser, mobile, etc.) - display alternatives on other parameters. This is not a new problem but an old problem, still getting worse. See #3 above for implementation examples via Apache .htaccess.

### *Exclusions by convention (i.e. robots.txt) or crawler-blocks by user-agent*

Exclusions by convention (robots.txt) or crawler-blocks by user-agent are not a new problem. It is an old problem that has been around since the early days of the Web. The /robots.txt is a de-facto standard, and is not owned by any standards body. There are two historical descriptions:

- 1994 A Standard for Robot Exclusion document.
- 1997 Internet Draft specification A Method for Web Robots Control

as well as the following external resource:

- HTML 4.01 specification, Appendix B.4.1

This example file excludes all bots but Google:

```
User-agent: Google
Disallow:

User-agent: *
Disallow: /
```

There is very little evidence that the problem is getting a lot worse or a lot better. Other changes far outstrip this issue as a leading concern, but it is important to track as an ongoing issue.

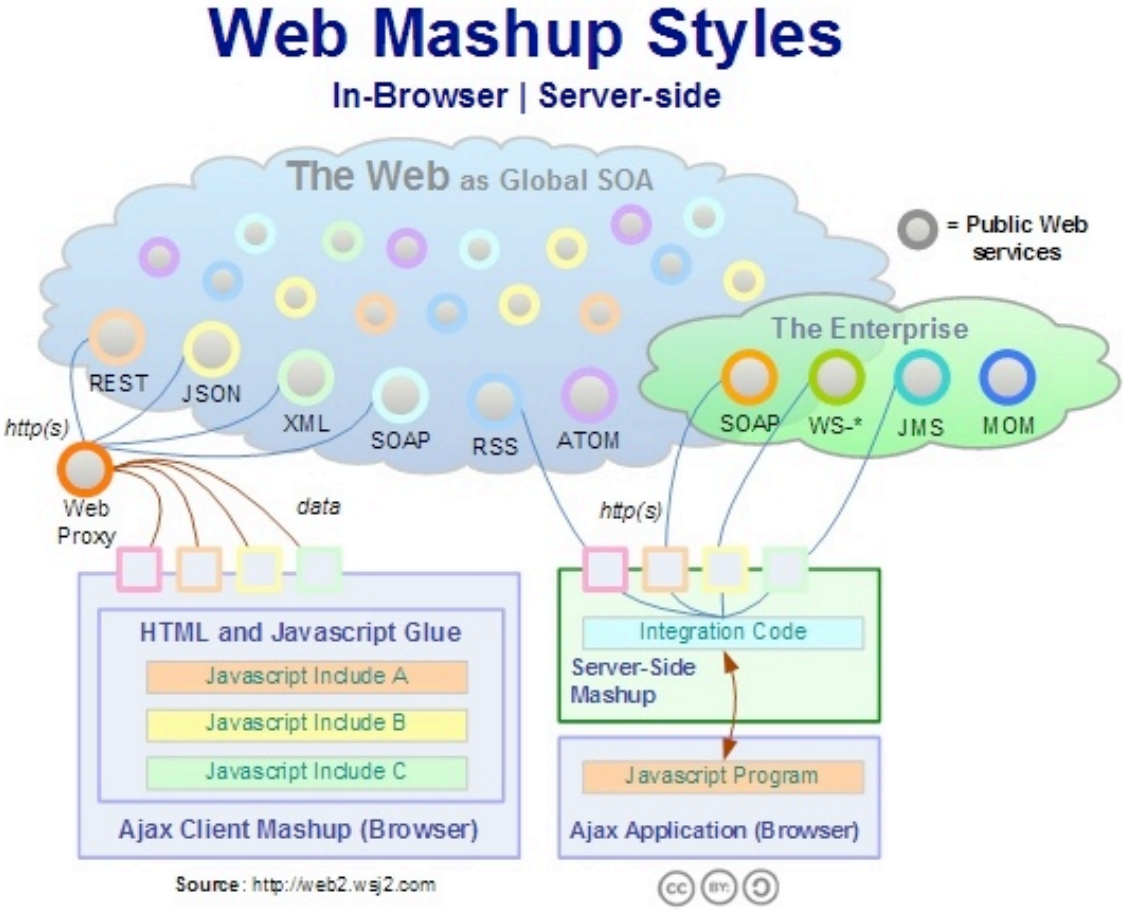
### *Exclusions by design (i.e. site architecture intended to inhibit crawling and indexing)*

Site publishers often intentionally attempt to thwart download and capture of the resources hosted “via” a site or online service. Exclusions by design (i.e. site

architecture intended to inhibit crawling and indexing) often are implemented by people with reasonable intentions but if not carefully monitored actions can have unintended consequences, e.g. when trying to stop third-parties from replicating/reoffering/rebranding their unique work attempts to collect resources can cause servers to crash, etc., if the bot filters aren't carefully coded.

### *Server side scripts & remote procedure calls*

Server side scripts & remote procedure calls: especially, when those trigger off any of the other challenges (very customized/volatile behavior), so that a single server request archived from collection time is just a tiny sliver (and not appropriate/representative) of functionality is not a new problem, but one that now describes 80+% of online resources, some to a greater or lesser degree than others. More generally what these implementations have in common is that rather than being listed in hypertext, the full variety of possible paths through a site are now often hidden in remote/opaque server-side code. Below are some visuals and code samples that illustrate aspects of these challenges.




Have you ever wanted to translate zip codes to Lat/Long to display in Google Maps? Here's the example to do it! [Don't forget that this code requires a REST service in Presto named "geonames":

```
http://ws.geonames.org/postalCodeSearch?postalcode=06700&country=MX.]

    <mashup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemalocation="http://www.jackbe.com/2008-03-01/EMMLSchema
../src/schemas/EMMLSpec.xsd"
        xmlns="http://www.jackbe.com/2008-03-01/EMMLSchema"
xmlns:macro="http://www.jackbe.com/2008-03-01/EMMLMacro"
        name="zip2lat">
      <operation
          name="invoke"> <!-- Here are our variables. Zipcode and Country are the variables in our
REST. They have default variables to avoid sending it empty and receive an unexpected result -->
        <input name="zipcode" type="string" default="06700"     />
        <input name="country" type="string" default="MX"     />
        <output name="result" type="document"></output>
        <!-- We will be saving the lat/lng in these variables to have them in our constructor. Perhaps
just a way to keep things organized since for later usage it could become a huge code if not
modularized -->
        <variables>
          <variable name="lat" type="string"></variable>
          <variable name="lng" type="string"></variable>
        </variables>
        <!-- Here we invoke the REST we published in service explorer and send the previous inputs to
it -->
        <invoke operation="getData" service="geonames" inputvariables="zipcode,country"
            outputvariable="preliminary"></invoke>
        <assign fromexpr="$preliminary//lat/string()" outputvariable="lat"></assign>
        <assign fromexpr="$preliminary//lng/string()" outputvariable="lng"></assign>
        <!-- Building the output -->
        <constructor outputvariable="result"     />
      </operation>
    </mashup>
```

**Src:** http://www.jackbe.com/enterprise-mashup/knowledge-base/code_sample/sample-mashup-code-mashup-convert-zipcode-latlong-data-rest-service

Multi-step processes with dependencies and interactions that are supported by both client and server (like workflows) are challenging to collect faithfully. Should archived workflows be representative vs. exact replicas? Is there a canonical user experience that can be captured or do we need to collect many individual experiences along a spectrum to create a representative sample? Are there synergies/lessons to be learned from preservation of scientific data sets and methods that can be applied to preservation of cultural applications and workflows online?

### *HTML5 "web sockets"*

HTML5 introduced "web sockets"- a platform for enabling “web applications”. The new publishing standards allow a client & server to keep a persistent 2-way communication channel over HTTP. This approach effectively codifies incremental

updates without page re-loads. It is expected that a lot of AJAX-based web "applications" (as opposed to documents) will migrate to a web socket-based implementation. Latest versions of all browsers support it, and the platform has achieved significant adoption amongst leading mobile devices and mobile service and app providers. On the flip side, there is better documentation of the resources used to assemble the end user experience. We are likely to have a better idea of what classical crawlers are missing/unable to capture during a typical crawl lifecycle.

Recommended background material:
HTML5: http://vimeo.com/25147748
HTML5 Websockets: http://www.websocket.org/
HTML5 Web RTC: http://www.webrtc.org/

Sample code pulled from O'Reilly, HTML5 for Publishers

```
window.addEventListener('load', eventWindowLoaded, false);
function eventWindowLoaded() {
    get_location();
    function get_location() {
        if (Modernizr.geolocation) {
            navigator.geolocation.getCurrentPosition(geolocate_story,
throw_error);
        } else {
            alert('Your browser/ereader does not support geolocation.
Sorry.');
        }
        function throw_error(position) {
            alert('Unable to geolocate you. Sorry.');
        }
    }
    function geolocate_story(position) {
        var geo_lat = position.coords.latitude;
        var geo_long = position.coords.longitude;
        // Get weather information
        $.ajax({
            type: 'GET',
            url: 'http://ws.geonames.org/findNearByWeatherXML?lat=' +
geo_lat + '&lng=' + geo_long,
            dataType: 'xml',
            success: function (weather_resp, xmlstatus) {
               var temperature_celsius =
$(weather_resp).find("temperature").text();
               if (temperature_celsius != "") {
                   // Weather temp data given in Celsius; convert to
Fahrenheit, because I'm American
                   var temperature_fahrenheit = 9/5*temperature_celsius
+ 32;
                   $('#weather_temp').text(temperature_fahrenheit);
               } else {
                   $('#weather_temp').text("TEMP NOT FOUND");
               }
            },
```

```
            error: function (xhr, status, error) {
                alert(error);
                $('#weather_temp').text("TEMP NOT FOUND");
            }
        })
        // Get full location information
        $.ajax({
            type: 'GET',
            url: 'http://ws.geonames.org/extendedFindNearby?lat=' +
geo_lat + '&lng=' + geo_long,
            dataType: 'xml',
            success: function (loc_resp, xmlstatus) {
                var city_name = $(loc_resp).find("placename").text();
                if (city_name != "") {
                    $('#city').text(city_name);
                } else {
                    $('#city').text("CITY NOT FOUND");
                }
                var street_address =
$(loc_resp).find("streetNumber").text() + " " +
$(loc_resp).find("street").text();
                if (street_address != "") {
                    $('#street_address').text(street_address);
                } else {
                    $('#street_address').text("ADDRESS NOT FOUND");
                }
            },
            error: function (xhr, status, error) {
                alert(error);
                $('#city').text("CITY NOT FOUND");
                $('#street_address').text("ADDRESS NOT FOUND");
            }
        })
    }
}
```

**Code Source:** https://github.com/sandersk/HTML5-for-Publishers/blob/a4595a87f9ff9ff7543bd1af2561ee1c2164bfcf/geolocation/geolocation-story.js

### *Mobile publishing*

Mobile publishing: the use of mobile phones to access the web is ever increasing, e.g. the stats that ~60%+ use of twitter is via mobile phones/devices. Many popular sites, especially social network sites/services, have a "mobile version" of their website, such as "m.facebook.com". This isn't new, but with the advent of the iPhone/Android, etc., it is much more wide spread than it was ten years ago. Unfortunately, not many organizations can/will be able to afford to crawl as multiple user agents to capture a range of end user experiences.

Most of the publishing methods and challenges listed above are in fact NOT novel. But, they are more broadly adopted and deployed today, and many more are included by default in site building templates and services, than they were ten or fifteen years ago - back when a website builder still had to code his/her own html and scripts to have anything more than a simple personal web page.

### *DRM - protected content (music, etc. )*

Even if content is captured with permission, the issues of being able to replay DRM protected content over time are significant. Ongoing access to ones music or films will come up for all individuals who purchase resources online. This is not a new problem at all, in fact it has been an issue since the very beginning of web publishing.

### *Paid content – "crawler + credit card" – hardly a match made in heaven*

This is not a new challenge. AOL can be said to have launched the trend in for-fee, "walled gardens". They've been around as subscription services since the Web began – initially as personals and genealogical services etc. However, is it appropriate for a crawl instance to mimic an end user down to the level of having an online financial presence? If so, you could configure a traditional crawler to enter the information into a form as long as it did not have the challenges of a dynamic form detailed above. One could also script a browser to do the same.

## Current Measures We Can Take to Try and Mitigate and/or Address Known Issues/Challenges

Many of the active measures that have been taken by individuals and organizations to address these challenges group together under the theme: incrementally making a classic, link-following crawler behave "more like" a web-browser with human-user.

### *Configure a classical crawler for capture*

E.g. most classical crawlers, including Heritrix (https://webarchive.jira.com/wiki/display/Heritrix/Heritrix), can handle cookies, logins to a URI that remains constant, downloadable scripts, video, audio, etc.

***Identify alternate targets that are "classical" crawler friendly*** (e.g. no script versions of a resource, etc. when avail/equivalent)

### *Extend a classical crawler*

- Add **browser based link extractors** (in use by IA),
- Add **browser emulators and scriptable browser instances** (in use by many IIPC members for for auto-qa, crawl analysis, and capture), some replay scenarios supported as well.
- **Let the crawler adopt a user profile or register the crawler as a user** (will want to modernize the login capabilities to support dynamic URIs for logins)

***Design new hybrid architecture/s*** that link multiple methods and resources together (in use for corporate, legal compliance web archiving services, e-discovery, bio science data capture, art catalog archiving, personal web archiving, and cultural heritage web archiving services, etc.)

***Use virtual machine instances*** to

- **Simplify proxy based replay via rendering tools** (like Wayback),
- **Constrain DNS traversal and content serving from the archive**,
- **Launch open source rich media players** to create a sense of native replay, etc. User might download the package locally. This would avoid interference with working applications on the desktop/laptop/device but might make it difficult to save a copy, print, etc. resources.

Compared to just enumerating the links on a page, these measures require more sophistication (and perhaps manual crawler config/training): specifying the exact

order in which steps must be executed; supplying form/auth data, enumerating (some subset of) all possible app gesture interactions (whose diversity may "combinatorially" explode, compared to link paths, potentially resulting in the identification of many possible paths that are uninteresting/repetitive).

Alternatively, the community could invest in "new" solutions:

***Coding new software solutions, alternate crawlers & replay tools*** (in use for corporate, legal compliance web archiving services, e-discovery, bio science data capture, art catalog archiving, personal web archiving, and cultural heritage web archiving services…)

***Adopting/integrating specialized tools for each file format/type*** (especially for rich media). There are many software packages we could use to capture data and write to archival files (WARCs), e.g. custom software for capturing streamed media, and to replay resources.

Compared to classical crawling and re-rendering, these approaches require assembly of appropriate subject matter expertise and management of complex systems integration efforts, business processes and workflows. And, the emerging solutions are never static, but must be constantly evolved as online content and services change and archival requirements shift alongside those changes.

Other alternatives borrow from traditional documentary practices:

***"Film" the user experience*** (record a video of complex server side environments)

***Take an image of the look and feel*** of a resource using one or more browser instances/emulators, and/or user profiles. Most participants liked this idea but expressed concerns regarding the scalability of the strategy to support Web scale capture.

***Deposit an instance of a "site" in a virtual instance with an archive –*** Most participants agreed that this approach held limited promise for addressing the current challenges. It might work as a complement to the traditional capture method for sites/resources that still adhere to publishing practices consistent with that model.

***Implement transactional web archiving on the server side –*** Too early to assess the viability of this approach – some research has begun at

LANL/ODU with the TWA project. Huge barrier to adoption since publishers/web site owners must implement code..

***Personal web archiving tools for distributed data capture?*** - how can we leverage the investment going into this space. Should we leverage commercial, personal web archiving tools or implement open source tools for individual archivists and define a standard mechanism for exchanging web data? Can we create P2P networks to support capture and preservation? Should we?

- Could the Locker project help (protect/control your own data)?
- Need to define data interchange standards at least for digital archives

Where do "Data liberation teams" fit? Do we hand them the tools and the repo space or should we leverage these networks for curation alone and assume we can surpass the technical challenges?

## Possible Areas to fund:

### *Development of an open source "archiving proxy" tool*

### *Publish the rate of occurrence of each web publishing practice/archiving issue we've identified*

We need to measure and publish the prevalence of each of these issues over time – rank the issues based on significance of impact in terms of end users accessing resources with these characteristics on a daily basis and volume of resources impacted over all in a tld. We should create an aggregate picture of what we discover as we crawl that can be shared across archiving institutions globally, i.e. a quantitative assessment of the problems and their evolution including the frequency and breadth of impact of each over time. This can be done without compromising rights issues or practices in individual states, regions, and countries.

### *Advocacy to publishing platform providers*

could make a significant impact on our ability to archive in future (e.g. Drupal, etc.)