# Chapter 4
# Hypermedia Types

**Mike Amundsen**

*The WWW is fundamentally a distributed hypermedia application.*

– Richard Taylor

*Hypermedia is defined by the presence of application control information embedded within, or as a layer above, the presentation of information.*

– Roy T. Fielding

**Abstract** It is generally understood that, in the REST architectural style, "hypermedia is the engine of application state" (Fielding 2000). But what does that really mean? What is hypermedia? Can it be identified within a resource representation? How can hypermedia be the "engine of application state?"

## Introduction

It is generally understood that, in the REST architectural style, "hypermedia is the engine of application state" (Fielding 2000). But what does that really mean? What is hypermedia? Can it be identified within a resource representation? How can hypermedia be the "engine of application state?"

In this chapter, a number of different notions of "hypermedia" along with a formal definition of "Hypermedia Type" will be presented. In addition, nine Hypermedia Factors (H-Factors) that can be found in resource representations are identified and examples of these factors are provided. Armed with these nine H-Factors, several registered media types are analyzed to determine the presence of these hypermedia elements and to quantify the hypermedia support native to these media types. Finally, a prototypical media type (*PHACTOR*) is defined and reviewed

M. Amundsen (✉)
Erlanger, KY 41018, USA
e-mail: mamund@yahoo.com

in order to show how H-Factors can be incorporated into a media type in order to produce a data format that can act as an engine of application state.

## The Various Roles of Hypermedia

The history of hyper[*text*|*data*|*media*][1] is long and varied. Although a full treatment of the history of hypermedia is beyond the scope of this chapter, several aspects will be covered here. The first three are (1) hypermedia as read-only links, (2) hypermedia as GUI controls for local applications, and (3) hypermedia as state transition controls for components in a widely distributed network. In addition, the notion of hypermedia as an essential part of distributed network architecture as well as the use of MIME Media Types in HTTP is covered. Finally, a definition of "Hypermedia Type" will be presented.

### *Hypermedia as Links*

The idea of hypermedia was given public voice by Vennevar Bush (1945) as a way to help researchers deal with what was perceived in the 1940s as an explosion of information. Bush described his idea for the "Memex" in a 1945 article, "As We May Think." In it he states, "The human mind ... operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain." (Bush 1945). He wanted to make it possible for information to be shared (using microfilm) and loaded into personal readers that could find links between subjects and allow the user to easily jump from one document to the next, following a single line of thought through a vast array of available content.

Decades later, in the 1974 self-published work, Computer Lib/Dream Machines (Nelson 1974), Theodor Nelson echoed Bush claiming "...writers do better if they don't have to write in sequence ... and readers to better if they don't have to read in sequence..." In this same work, Nelson coins the terms "hypertext" and "hypermedia" saying "By 'hypertext,' I mean non-sequential writing – text that branches and allows choices to the reader, best read at an interactive screen. As popularly conceived, this is a series of text chunks connected by links which offer the reader different pathways." (Nelson 1974).

In both these examples, hypermedia is thought of as a way to provide links between related materials and enable readers to move freely along these related paths. Hypermedia, in this case, is limited to a read-only experience meant for enriching reading, discovery, and comprehension of text.

[1]The words "hypertext", "hyperdata" and "hypermedia" all have seen active use; sometimes to mean different things. In this chapter, the word "hypermedia" indicates the general concept of links that provide 'jumps' or branches in text or any visual display. Therefore, throughout the rest of this chapter "hypermedia" will be used exclusively.

## *Hypermedia as GUI Controls*

While the movement to enable improving the use-ability of text was underway, a second line of thought was also taking shape. That of using hypermedia as a way to control the location and retrieval of data for display to the user: hypermedia as a feature of graphical use interfaces.

As a radar station operator in the Philippines during World War II, Doug Engelbart happened upon Vannevar Bush's magazine article and was fascinated with the idea of the "Memex." Years later, Engelbart would publish "Augmenting Human Intellect: A Conceptual Framework" (Engelbart 1962) where he laid out his interpretation of Bush's vision.

"Most of the structuring forms ... stem from the simple capability of being able to establish arbitrary linkages between different substructures, and of directing the computer subsequently to display a set of linked substructures with any relative positioning we might designate among the different substructures." (Engelbart 1962).

By 1968, Engelbart had developed the NLS (oN-Line System) for sharing research information. His demonstration included not just a computer information system capable of supporting links between items, but the first "mouse" pointing device that could be used to actuate those links on screen. A video demonstration of this early hypertext system is still available for viewing (Engelbart 1968).

Later, in 1987, Jeffrey Conklin published "Hypertext: An Introduction and Survey" (Conklin 1987) which described hypertext as "...a computer-supported medium for information in which many interlinked documents are displayed with their links on a high-resolution computer screen." Conklin's work focuses on the role hypertext plays in graphical user interfaces (GUIs) and their influence on user interfaces in general. Conklin also compares editing environments for hypertext content.

Additional development of the personal computer through the 1980s and early 1990s introduced more display options and additional ways to express hypermedia links. The concept of hypermedia was expanded to include actuating interface controls such as selectors and buttons and spawned an emphasis on visual controls users can activate at any time. Hypermedia had become more than linking text, it was also a visual "affordance" to animate user displays.

## *Hypermedia as Application Controls*

At the same time personal computer displays were providing more graphical controls, early versions of the World Wide Web appeared. In 1991, Tim Berners-Lee's WWW was available and, by 1993 the NSCA Mosaic Web Browser had become the popular graphical user interface for the WWW.

Along with the assumed graphical link elements that allowed WWW users to "jump" from one document to the next (or within documents), Web browsers had the ability to render images within the current document and the ability to send content to the server using link templates implemented as forms with input elements users could fill in and submit.

With the introduction of forms, hypermedia was no longer limited to static, read-only experiences. Documents could be created that allowed users to send data as well as find and retrieve it. HTML, the *de facto* document format for the WWW, allowed for the inclusion of application controls along with human-readable text.

Years later, in a slide presentation to ApacheCon 2005 Roy Fielding would describe this use of hypertext: "When I say hypertext, I mean the simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions." (Fielding 2008).

## *Hypermedia as Architecture*

The notion of hypermedia as more than text, more than data, but also application controls that allow users to make choices along the way is an important requirement for RESTful implementations over distributed networks. In addition to representing the state of the requested resource, Hypermedia documents contain the affordances for changing the state of the application. It is the hypermedia that makes state transitions possible. This alters the role of server responses from simple data replies to that of an essential part of the network architecture.

The idea that data itself (whether simple state information or hypermedia controls) can be part of the network architecture was articulated by Fielding (2000) as "...the nature, location, and movement of data elements within the system is often the single most significant determinant of system behavior." Even more directly, Fielding continues "The nature of the data elements within a network-based application architecture will often determine whether or not a given architectural style is appropriate." Finally, Fielding identifies hypermedia specifically as the "engine of application state."

## *MIME Types, HTTP, and Hypermedia Types*

As mentioned in Fielding's 2001 dissertation (Fielding 2000), "HTTP inherited its syntax for describing representation metadata from the Multipurpose Internet Mail Extensions (MIME)." For this reason, RESTful implementations over HTTP are tied to using MIME Media Types for representing requests and responses. The official registry for MIME media types at the Internet Assigned Numbers Authority (IANA), contains hundreds of media type formats and descriptions. It would seem anyone setting out to create a RESTful implementation would have no trouble finding a wide range of suitable media types for handling their hypermedia resource representations.

However, a cursory review of solutions on the Web reveals that a relatively small number (not including binary representations for images, archive formats, etc.) of registered media types are consistently used as resource representation formats. Media types from that list that are widely supported are HTML (Raggett et al. 1999), Atom (Nottingham et al. 2005), XML (Bray et al. 2008), and JSON (Crockford 2006). Why is this the case?

The reason for favoring these few types is not merely historical priority. HTML has been around since the very start of the WWW and Atom earned Standards Track status in 2005. XML (first approved in 1998) and JSON (approved in 2006) are also often-used structured data formats. Apart from varying origin dates, these four media types have another defining characteristic worth noting. HTML and Atom include, as part of their format, well-defined link elements with clearly-associated protocol semantics. On the other hand, XML and JSON have no such defined native elements.

Having protocol semantics (e.g. HTTP GET, POST, etc.) defined within, and bound to elements of, the media type is an important distinction. Media types that share this trait are uniquely capable of enabling Fielding's "engine of application state." They are *hyper*media types. This discovery leads to a simple, but useful definition of "Hypermedia Type":

*A Hypermedia Type is a media type that contains native hyper-linking elements that can be used to control application flow.*

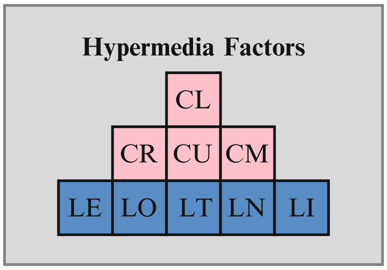
## *Summary*

This section focused on various views of hypermedia itself; from read-only links to application controls. This last aspect of hypermedia – as a way to control applications through state transitions – has an important relation to architectural styles for distributed networks themselves. When responses are expressed as hypermedia documents, these responses are more than just data, they are also part of the network architecture.

Finally, a formal definition of the term "Hypermedia Type" was introduced.

## Nine Hypermedia Factors

This section identifies nine factors native within a media type that can be used to support hypermedia behaviors. These nine "H-Factors" can be expressed in various ways, depending on the media type itself. However, no matter what actual elements or attributes are used to express these factors, the factors themselves remain essentially the same across all hypermedia types.



this figure will be printed in b/w

Each H-Factor identifies a clear hypermedia interaction between client and server. To this end, the H-Factors are divided into two distinct groups: “link” factors (`LO, LE, LT, LI, LN`) and “control data” factors (`CR, CU, CM, CL`). The five “link” factors denote specific linking interactions between parties: Outbound, Embedded, Templated, Idempotent, and Non-Idempotent, respectively. The remaining four “control data” factors provide support for customizing metadata details (e.g. HTTP Request Headers) of the hypermedia link interaction: Reads, Updates, Method, and Link Annotations.

| Hypermedia factors | | |
|---|---|---|
| Links | LO | Outbound Links |
| | LE | Embed Links |
| | LT | Templates Links |
| | LN | Non-Idempotent Links |
| | LI | Idempotent Links |
| Control data | CR | Read Controls |
| | CU | Update Controls |
| | CM | Method Controls |
| | CL | Link Annotation Controls |

It should be noted that, to this author’s knowledge there is no single registered media type that contains all nine of these factors. In fact, some media types contain none at all, some contain just one or two, etc.

Below is a list of the nine H-Factors, their descriptions and examples from well-known, registered media types.

## *Embedded Links: LE*

The LE factor indicates to the client application that the accompanying URI should be de-referenced using the application-level protocol’s read operation (e.g. `HTTP GET`) and the resulting response should be displayed within the current output window. In effect, this results in merging the current content display with that of the content at the "other end" of the resolved URI. This is sometimes called "transclusion."

A typical implementation of the LE factor is the `IMG` markup tag in HTML:

```
<img src="..." />
```

In the above example, the URI in the `src` attribute is used as the read target and the resulting response is rendered "inline" on the Web page.

In XML, the same LE factor can be expressed using the `x:include` element.

```
<x:include href="..." />
```

## *Outbound Links: LO*

The LO factor indicates to the client application that the accompanying URI should be de-referenced using the application-level protocol's read operation and the resulting response should be treated as a complete display. Depending on additional control information, this may result in replacing the current display with the response or it may result in displaying an entirely new viewport/window for the response. This is also known as a "traversal" or "navigational" link.

An example of the LO factor in HTML is the `A` markup tag:

```
<a href="...">...</a>
```

In a common Web browser, activating this control would result in replacing the current contents of the viewport with the response. If the intent is to indicate to the client application to create a new viewport/window in which to render the response, the following HTML markup (or a similar variation) can be used:

```
<a href="..." target="_blank">...</a>
```

## *Templated Links: LT*

The LT factor offers a way to indicate one or more parameters that can be supplied when executing a read operation. Like the LE and LO factors, LT factors are read-only. However, LT factors offer additional information in the message to instruct clients on accepting additional inputs and including those inputs as part of the request.

The LT element is, in effect, a link template. Below is an example LT factor expressed in HTML using the `FORM` markup tag:

```
<form method="get" action="http://www.example.org/">
  <input type="text" name="search" value="" />
  <input type="submit" />
</form>
```

HTML clients understand that this LT requires the client to perform URI construction based on the provided inputs. In the example above, if the user typed “hypermedia” into the first `input` element, the resulting constructed URI would look like this:

```
http://www.example.org/?search=hypermedia
```

The details on how link templates (LT) are expressed and the rules for constructing URIs depends on the documentation provided within the media type itself.

Templated links can also be expressed directly using tokens within the link itself. Below is an example of a templated link using specifications from the URI Template I-D (Gregorio et al. 2010):

```
<link href="http://www.example.org/?search={search}"/>
```

## *Non-Idempotent Links: LN*

The LN factor offers a way to send data to the server using a non-idempotent “submit.” This type of request is implemented in the HTTP protocol using the `POST` method. Like the LT factor, LN can offer the client a template that contains one or more elements that act as a hint for clients. These data elements can be used to construct a message body using rules defined within the media type documentation.

The HTML `FORM` element is an example of a non-idempotent (LN) factor:

```
<form method="post" action="http://example.org
/comments/">
  <textarea name="comment"></textarea>
  <input type="submit" />
</form>
```

In the above example, clients that understand and support the HTML media type can construct the following request and submit it to the server:

```
POST /comments/ HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded
Length: XX

comment=this+is+my+comment
```

It should be noted that the details of how clients compose valid payloads can vary between media types. The important point is that the media type identifies and defines support for non-idempotent writes.

## *Idempotent Links: LI*

The LI factor provides a way for media types to define support for idempotent submits. These types of requests in the HTTP protocol are supported using the `PUT` and `DELETE` methods. While HTML does not have direct support for idempotent submits within markup (e.g. `FORM method="PUT"`), it is possible to execute idempotent submits within an HTML client using downloaded code-on-demand.

Below is an example idempotent link factor (LI) expressed using Javascript:

```
<script type="text/javascript">
  function delete(id)
  {
    var client = new XMLHttpRequest();
    client.open("DELETE", "http://example.org
    /comments/"+id);
  }
</script>
```

The Atom media type implements the LI factor using a `link` element with a relation attribute set to "edit" (`rel="edit"`):

```
<link rel="edit" href="http://example.org/edit/1"/>
```

Clients that understand the Atom specifications know that any link decorated in this way can be used sending idempotent requests (`HTTP PUT, HTTP DELETE`) to the server.

## *Read Controls: CR*

One way in which media types can expose control information to clients is to support manipulation of control data for read operations. The HTTP protocol (Fielding et al. 1999) identifies a number of HTTP Headers for controlling read operations. One example is the `Accept-Language` header. Below is an example of XInclude (Marsh et al. 2006) markup that contains a custom `accept-language` attribute:

```
<x:include
  href="http://www.exmaple.org/newsfeed"
  accept-language="da, en-gb;q=0.8, en;q=0.7"
/>
```

## *Update Controls: CU*

Support for control data during send/update operations (CR) is also possible. For example, in HTML, the `FORM` can be decorated with the `enctype` attribute.

The value for this attribute is used to populate the `Content-Type` header when sending the request to the server.

```
<form method="post"
  action="http://example.org/comments/"
  enctype="text/plain">
  <textarea name="comment"></textarea>
  <input type="submit" />
</form>
```

In the above example, clients that understand and support the HTML media type can construct the following request and submit it to the server:

```
POST /comments/ HTTP/1.1
Host: example.org
Content-Type: text/plain
Length: XX

this+is+my+comment
```

## *Method Controls: CM*

Media types may also support the ability to change the control data for the protocol method used for the request. HTML exposes this CM factor with the `method` attribute of the `FORM` element.

In the first example below, the markup indicates a send operation (using the POST method). The second example uses the same markup with the exception that the GET method is indicated. This second example results in a read operation.

```
<form method="post" action="..." />
  <input name="keywords" type="text" value="" />
  <input type="submit" />
</form>
```

```
<form method="get" action="..." />
  <input name="keywords" type="text" value="" />
  <input type="submit" />
</form>
```

## *Link Controls: CL*

In addition to the ability to directly modify control data for read and submit operations, media types can define CL factors which provide inline metadata for the links themselves. Link control data allows client applications to locate and understand the meaning of selected link elements with the document. These CL factors provide a way for servers to "decorate" links with additional metadata using an agreed-upon set of keywords.

For example, Atom (Nottingham et al. 2005) documentation identifies a list of registered Link Relation Values (IANA Protocol Registries 2011) that clients may encounter within responses. Clients can use these link relation values as explanatory remarks on the meaning and possible uses of the provided link. In the example below, the Atom `entry` element has a `link` child element with a link relation attribute set to "edit" (`rel="edit"`).

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-
      80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>John Doe</name></author>
  <content>Some text.</content>
  <link rel="edit" href="http://example.org/edit/1"/>
</entry>
```

Clients that understand the Atom and AtomPub (Gregorio et al. 2007) specifications know (based on the documentation) that any link decorated in this way is the link to use when sending idempotent submits (`HTTP PUT, HTTP DELETE`) to the server.

Another example of using CL factors is HTML's use of the `rel="stylesheet"` directive (see below).

```
<link rel="stylesheet" href="..." />
```

In the above example, the client application (Web browser) will use the URI supplied in the `href` attribute as the source of style rendering directives for the markup in the HTML document.

## *Summary*

This section identified nine Hypermedia Factors; one or more of which can be found in a media type document. The presence of these factors within the media type definition mark it as a hypermedia type and indicate support for various protocol-level hypermedia semantics. MIME media types that contain one or more of these factors promote RESTful implementations by making it possible to include application controls within the requests and responses. These hypermedia application controls are the elements of the message advance application flow.

## Analyzing Media Types

Once a set of Hypermedia Factors have been defined, it is a simple matter to review any MIME media type and identify those H-Factors in the selected media type. By cataloging the H-Factors in a given type, architects and implementors can make assessments about the fitness of a particular media type for the intended implementation.

For example, an implementation that must support HTTP PUT and DELETE (H-Factor LI) should not rely solely on the HTML media type for resource representations since HTML has no native support for LI elements. Or, to use another example, if the proposed implementation requires support for templated links (LT), the Atom media type may not be the best selection for all use cases. However, by matching the hypermedia needs of the RESTful implementation to the H-Factors found in existing media types, a "best fit" of one or more media types can be identified for each use case.

Below are sample analyses of some registered MIME media types (URI List (Mealling et al. 1999), SVG (Dahlstrm et al. 2011), HTML (Raggett et al. 1999) and Atom (Nottingham et al. 2005).[2] In the interest of space, these media types are not exhaustively reviewed, but example elements are identified that meet the specifications of one or more of the H-Factors outlined previously in this chapter. This is done to give a general guide to the process of analyzing existing media types for the appearance of H-Factors as native elements.

### *Media Types Void of H-Factors*

Some well-known media types are not covered here including XML (Bray et al. 2008) and JSON (Crockford 2006). These two (and similar ones) have been left out for an important reason: they contain no native H-Factors as part of their definition. In other words, these media types exhibit no defined elements capable of expressing any of the previously identified H-Factor links (LO, LE, LT, LI, LN) or control data (CR, CU, CM, CL).[3]

[2]It should be noted that these are not the only media types that warrant hypermedia analysis. They are also not selected here as excellent examples of Hypermedia Types, but merely as familiar media types worthy of review.

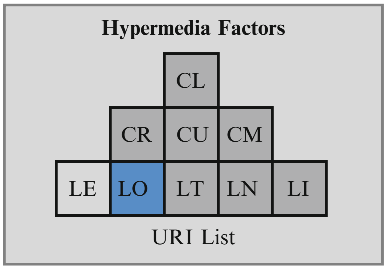
[3]While it is true that media types such as XML and JSON *allow* designers to define link and control elements *using* the basic elements of that media type, this does not qualify as providing native support for H-Factors.

## *URI List*

A very simple example of a hypermedia type is the `text/uri-list` media type (Mealling et al. 1999). This media type consists of nothing more than a list of URIs:

```
# urn:isbn:0-201-08372-8
http://www.huh.org/books/foo.html
http://www.huh.org/books/foo.pdf
ftp://ftp.foo.org/books/foo.txt
```

this figure will be printed in b/w

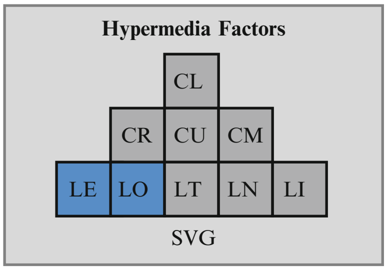


This media type is designed to convey a list of one or more URIs that can be resolved and/or processed by the recipient. For the sake of analysis, this media type provides support for the LO (Outbound Link) Hypermedia Factor. It might be argued that the URIs could be treated by recipients as LE (Embedded Links) (e.g. image links merged into an existing document), but most of the suggested uses in documentation point to de-referencing and processing each URI in turn rather than using the list to produce a single composite document.

## *SVG*

Similar to the `text/uri-list` media type, the SVG media type (Dahlstrm et al. 2011) (`application/svg+xml`) exhibits support for a limited set of Hypermedia Factors. In this case, they are (1) the LO factor and (2) the LE factor.

this figure will be printed in b/w



The most common example of LO (Outbound Link) is the `A` element:

```
<a xlink:href="http://www.example.org">
  <ellipse cx="2.5" cy="1.5" rx="2" ry="1"
  fill="red" />
</a>
```

A common example of LO (embedded link) is the `image` element:

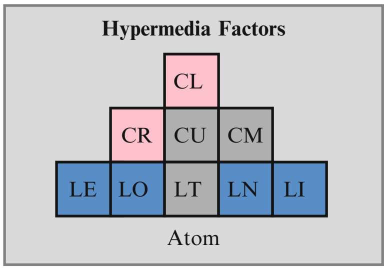
```
  <image x="200" y="200" width="100px" height="100px"
    xlink:href="myimage.png">
    <title>My image</title>
  </image>
```

While the SVG media type has a number of elements and attributes that support some form of URIs, all of these elements exhibit either the LO or LE H-Factors. Specifically, the SVG media type does not provide native support for LT (Templated Links), LI (Idempotent Link), or LN (Non-Idempotent Link) H-Factors.

## *Atom*

The Atom media type profile is defined by two specifications Atom (Nottingham et al. 2005) and AtomPub (Gregorio et al. 2007). There are three registered media types associated withAtom: `application/atom+xml`, `application/atomcat+xml`, and `application/atomsvc+xml`. The Atom specification outlines the message format and elements for the `application/atom+xml` media type. The AtomPub specification covers the details for the remaining two media types as well as the read/write semantics for all three media types.

this figure will be printed in b/w



The primary hypermedia element in the Atom media type family is the `atom:link` element:

```
<link href="..." rel="..." hreflang="en" />
```

The `link` element show above supports LO, CR, and CL H-Factors. The Atom semantic model also supports LI and LN H-Factors by identifying markup elements within the response that have special significance.

For example, non-idempotent writes can be used to add new `entry` elements to the collection. Clients are instructed to locate the `atom:link` element associated with the root of the response which is marked with `rel="self"`. This element's `href` is the "Collection URI" and is the target URI for executing non-idempotent writes to the collection:

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <link href="..." rel="self" />
  ...
</feed>
```

Idempotent writes (including updates and deletions) are indicated using the `rel="edit"` attribute on a `link` element that is the child of an `entry` element (see example below).

```
<entry>
  <link href="..." rel="edit" />
  ...
</entry>
```

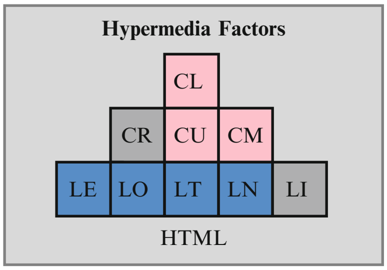
There are a handful of other elements in the Atom media type family that support both LO and LE H-Factors including:

```
atom:collection (LO)
atom:content (LO,LE)
atom:generator (LO)
atom:icon (LE)
atom:logo (LE)
atom:uri (LO)
```

It should be noted that the Atom media type family uses documentation convention to communicate the details of a valid write payload (LI, LN) and does not include these details within the response message itself. Also, Atom has no native support for Templated Links (LT).

## *HTML*

The HTML (Raggett et al. 1999) media type offers a wide range of support for H-Factors including: LO, LE, LT, LN and CU, CM, CL. The only H-Factors not supported in the HTML media type are LI (idempotent writes) and CR (control data for reads).



this figure will be printed in b/w

Simple outbound and embedded links are handled, for example, using the `A` and `IMG` tags respectively:

```
<a href="...">...</a>
<img src="..." />
```

HTML is the only media type covered in this chapter that supports Templated Links (LT). Templated links are similar to the LO H-Factor except that LT elements allow for URI variables to be defined by the sender and the values for these variables to be supplied by the client. In the HTML media type, this is accomplished using the `FORM` element with the `method="get"` attribute. The URI for the operation is found in the `action="..."` attribute. Below is an example:

```
<form method="get" action="http://example.org
/products" />
  <input type="text" name="color" value="" />
  <input type="text" name="size" value="" />
  <input type="submit" />
</form>
```

The HTML documentation instructs clients to use the inputs to amend the supplied URI before submitting the request to the server. Using the example above, and assuming input values, the constructed URI would look like the following:

```
http://example.org/products?color=red&size=large
```

Non-idempotent writes (i.e. the LN H-Factor) are supported using almost an identical markup as that seen for link templates. The only difference is the use of the `method="post"` attribute setting. For example, the same FORM element shown earlier can be used to write data to the server:

```
<form method="post" action="http://example.org/
  products"
  enctype="application/x-www-form-urlencoded" />
  <input type="text" name="color" value="" />
  <input type="text" name="size" value="" />
  <input type="submit" />
</form>
```

In the above case, HTML documentation instructs clients to use the inputs to construct a message body using (the default) `application/x-www-form-urlencoded` media type format. The resulting HTTP request sent to the server is:

```
POST /products HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded
Length: XX

color=red&size=large
```

It should be noted that, unlike the Atom media type which relies on documentation to tell clients how to compose a valid write message, the HTML media type allows servers to send clients write templates within the response and to use that template to compose a valid write request.

As seen in the two previous examples (LT and LN), HTML supports customizing control data protocol methods used for requests (CM) through the use of the `method` attribute of the `FORM` element and support for customizing control data for updates (CU) using the `enctype` attribute of the same element.

HTML also supports customizing control data for links (CL) using the `rel` attribute for the `link` tag. In the following example, the HTML client will use the response for the URI as style rules for adjusting the rendering of the content.

```
<link rel="stylesheet" href="... type="text/css" />
```

## *Summary*

This section reviewed a representative sample of registered MIME media types and subjected them to a simple analysis in order to identify native elements within the media type that express one or more of the identified H-Factors. The analysis was cursory (in order to save time and space), but the reader should now have a good idea of how this can be undertaken for any registered (or proposed) media type design.

## *PHACTOR*: A Prototypical Hypermedia Type

Armed with the knowledge of the nine Hypermedia Factors and experience analyzing existing media types, it is possible to construct a prototypical media type that illustrates each of the H-Factors defined in this chapter.

The goal of this exercise is to create an "illustration" media type that can be used as a guide when setting out to analyze other media types or as an aide for those wishing to design their own Hypermedia Type. This prototypical media type contains all the identified H-Factors (LO, LE, LT, LN, LI and CR, CU, CM, CL).

What follows are the specifications for a media type called *Prototypical Hypermedia Application Controls for Text-Oriented Representations* or *PHACTOR*[4]. Like HTML, PHACTOR is a hypermedia type designed to support rendering and layout of text-based documents. For this reason, the reader will find many similarities between HTML and PHACTOR.

## *PHACTOR Layout Elements*

*PHACTOR* is an XML-based media type used for representing simple text and a basic set of application controls. The main layout of a valid *PHACTOR* document is:

```
<document>
  <meta />
  <content />
</document>
```

The `meta` section can optionally hold one each of the following: `title, updated, author`.

```
<document>
  <meta>
    <title>H-Factor Sample</title>
    <updated>2010-06-15</updated>
    <author>MikeA</author>
  </meta>
  <content />
</document>
```

The `content` section can optionally hold one or more of the following elements (shown here in parent-child order): `section, para, text`. The `title` element may also appear as the first child of a `section` element. Also, the `eol` ('end-of-line') element can be used to create line breaks within a block of text.

```
<document>
  <meta>
    <title>H-Factor Sample</title>
    <updated>2010-06-15</updated>
    <author>MikeA</author>
  </meta>
```

[4]At the time of this writing, the *PHACTOR* media type has been submitted to the IANA Media Type registry with the `application/vnd.phactor+xml` MIME type identifier.

```
    <content>
      <section>
        <title>An implementation</title>
        <para>
          <text>
             This is a trivial hypermedia type
             implementation.<eol/>
             This is a new line in the document.
          </text>
        </para>
      </section>
    </content>
  </document>
```

### *PHACTOR Link Elements*

In addition to simple text and layout elements, the *PHACTOR* media type supports the following link elements: `LO, LE, LT, LN, LI`. These link elements can appear as child elements of the following elements: `content, section, para`. Also, the `LT, LN, LI` elements may have one or more optional `data` child elements. The `data` can be used to create link templates (`LT`) and to populate `LN` and `LI` representations to send to the server.

Below are examples of each of the link elements. Note that both the `LT` and `LN` elements use `data` child elements to define templates.

```
<LO href="http://example.org" label="example.org" />

<LE href="http://example.org/images/photo.jpg"
label="Photo" />

<LT href="http://example.org/search" >
  <data name="keyword" label="Search" />
</LT>

<LN href="http://example.org/comments/" label="Add
Your Comment">
  <data name="nickname" label="Nickname" />
  <data name="comment" label="Comment" />
</LN>

<LI CM="delete" href="http://example.org/comments
/123" label="Delete Comment"/>
```

## PHACTOR Control Data Elements

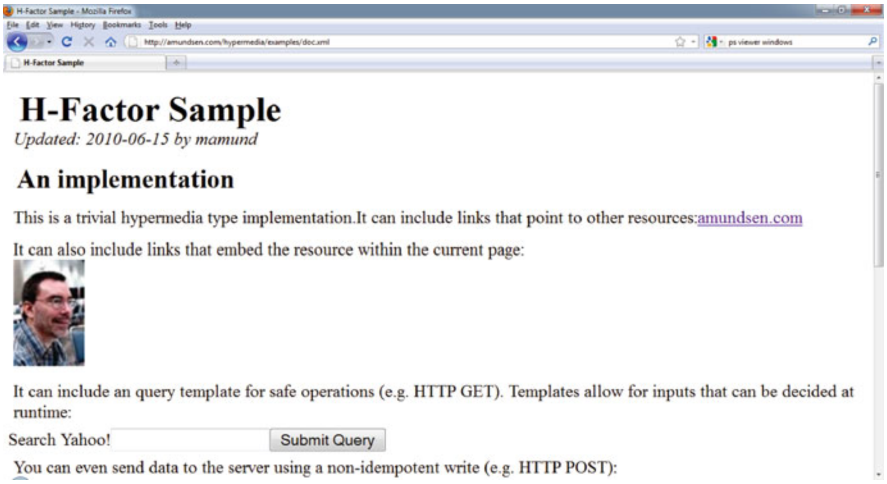
The *PHACTOR* media type supports optional control data for linking elements `CR, CU, CM, CL`. The `LE` element supports the `CR="[acceptLanguage]"` attribute to allow customizing the `Accept-Language` header of the request. The `LI` element supports the `CM="[protocolMethod]"` and `CU="[contentType"]` attributes to allow customizing the request with the protocol method and content type string respectively.

All five link types (`LO, LE, LT, LI, LN`) support the use of the `CL = "[linkRelationValue]"` attribute in order to decorate hypermedia links with additional metadata. Valid values for this attribute can be any registered link relation value or any fully-qualified unique URI [per RFC5988 (Nottingham 2010)]. It is up to the client application to determine the importance and meaning of this value.

## A Complete PHACTOR Document

Below is a complete sample *PHACTOR* document along with a screen-shot sample rendering of the same document in a Web browser.

```
<document>
  <meta>
    <title>H-Factor Sample</title>
    <updated>2010-06-15</updated>
    <author>MikeA</author>
  </meta>
  <content>
    <section>
      <title>An implementation</title>
      <para>
        <text>This is a trivial hypermedia type
        implementation.</text>
        <text>It can include links that point to other
        resources:</text>
        <LO CL="document" href="http://amundsen.com"
        label="amundsen.com" />
      </para>
      <para>
        <text>
          It can also include links that embed the
          resource
          within the current page:
        </text>
        <eol />
        <LE CL="document" href="http://amundsen.com/
        images/mca.jpg" label="mamund" />
      </para>
      <para>
        <text>
            It can include a query template read
            operations (e.g. HTTP GET).
            Templates allow for inputs that can be
            decided at runtime:
        </text>
      </para>
      <LT CL="search" href="http://search.yahoo.com
      /search" >
        <data name="p" label="Search Yahoo!" />
      </LT>
    </section>
  </content>
</document>
```


H-Factor Sample - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://amundsen.com/hypermedia/examples/doc.xml
ps viewer windows
H-Factor Sample
Submit Query

H-Factor Sample

Updated: 2010-06-15 by mamund

An implementation

This is a trivial hypermedia type implementation.It can include links that point to other resources:amundsen.com

It can also include links that embed the resource within the current page:

It can include an query template for safe operations (e.g. HTTP GET). Templates allow for inputs that can be decided at runtime:

Search Yahoo!

You can even send data to the server using a non-idempotent write (e.g. HTTP POST):

## *Rendering PHACTOR Documents*

Since the *PHACTOR* media type is based on XML, it is relatively easy to load, parse, and render using modern Web browsers. All modern Web browsers support client-side XSLT transformations. For the *PHACTOR* media type, each response can be accompanied by an XSLT stylesheet directive like the one below:

```
<?xml-stylesheet type="text/xsl" href="phactor.xsl"?>
```

This directive can be used by the browser client to transform the XML response into valid XHTML that can be rendered by the client application. Also, since *PHACTOR* has support for Idempotent Links (LI), a full-featured Web browser implementation requires use of the `XMLHttpRequest` API (van Kesteren 2010).

Below is a partial listing of the transformation document:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

<xsl:output method="html"
  media-type="text/html"
  doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
  doctype-system="DTD/xhtml1-strict.dtd"
  cdata-section-elements="script style"
  indent="yes"
  encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <html>
      <head>
        <xsl:apply-templates select="//meta" mode=
        "head" />
        <link href="doc.css" rel="stylesheet" type="
        text/css" />
      </head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>

  <!-- head items -->
  <xsl:template match="meta" mode="head">
    <title><xsl:value-of select="title" /></title>
    <meta name="updated" content="{updated}" />
    <meta name="author" content="{author}" />

  </xsl:template>

  ...
```

## *Summary*

In this chapter the varying role of hypermedia was reviewed including usage to express read-only links, act as GUI controls, and as a way to provide support for state transitions between distributed components. Hypermedia was also viewed from the perspective of network architecture itself.

Special attention was given to MIME Media Types and their use in HTTP in order to carry hypermedia information. It was observed that only a subset of MIME media types exhibit native hypermedia elements and these media types were used as the basis for a formal definition of "Hypermedia Type."

Nine native Hypermedia Factors (H-Factors) were introduced to show how media types can express application controls for various state transition activities in a RESTful implementation. In addition, several well-known registered media types were analyzed for the presence of H-Factors in order to quantify support for hypermedia in each media type.

Finally, a prototypical hypermedia type (*PHACTOR*) was presented to illustrate how media types can be designed using H-Factors. A sample page was presented and a sample *PHACTOR* user agent implementation based on a modern Web browser's support for XSLT and XMLHttpRequest was discussed.

## References

Bray, Tim, Ed. et al., *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, http://www.w3.org/TR/xml/ (2008)

Bush, Vannevar, *As We May Think*, Atlanic Magazine, July 1945

Conklin, Jeff, *Hypertext: An Introduction and Survey* in IEEE Computer, 20(9), 17–41, September 1987

Crockford, Douglas, *The application/json Media Type for JavaScript Object Notation (JSON)*, http://tools.ietf.org/html/rfc4627 (2006)

Dahlstrm, Erik, et al., *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*, http://www.w3.org/TR/SVG/ (2011)

Engelbart, Douglas, *Augmenting Human Intellect: A Conceptual Framework*, October 1962

Engelbart, Douglas, *The Demo*, http://sloan.stanford.edu/mousesite/1968Demo.html (1968)

Fielding, Roy Thomas, *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000

Fielding, Roy Thomas, *REST APIs must be Hypertext-driven*, http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-drivencomment-718 (2008)

Fielding, Roy Thomas, Ed. et al., *Hypertext Transfer Protocol – HTTP/1.1*, http://tools.ietf.org/html/rfc2616 (1999)

Gregorio, J., Ed. et al., *URI Template*, http://tools.ietf.org/html/draft-gregorio-uritemplate-04 (2010)

Gregorio, J., Ed. et al., *The Atom Publishing Protocol*, http://tools.ietf.org/html/rfc5023 (2007)

IANA Protocol Registries, *Link Relations*, http://www.iana.org/assignments/link-relations/ (2011)

van Kesteren, Anne, *XMLHttpRequest*, http://www.w3.org/TR/XMLHttpRequest/ (2010)

Marsh, Jonathan, et al., *XML Inclusions (XInclude) Version 1.0 (Second Edition)*, http://www.w3.org/TR/xinclude/ (2006)

Mealling, M. et al., *URI Resolution Services Necessary for URN Resolution*, http://tools.ietf.org/html/rfc2483 (1999)
Nelson, Theodor H., *Literary Machines*. Swarthmore, Pa.: Self-published (1974)
Nottingham, M., *Web Linking*, http://tools.ietf.org/html/rfc5988 (2010)
Nottingham, M., Ed. et al., *The Atom Syndication Format*, http://tools.ietf.org/html/rfc4287 (2005)
Raggett, Dave, Ed. et al., *HTML 4.01 Specification*, http://www.w3.org/TR/html401/ (1999)