IHTML 2: Design and Implementation

Gord Brown
University of Victoria
e-mail: gdbrown@acm.org

May, 1998

Abstract

A new version of Intensional HTML has been developed, with improved reliability, performance, and usability. This implementation is an enhancement to the Apache (tm) WWW server. Features include user-specified dimension names, dimension variables, version-based “case” statements in documents, and utilities for constructing an IHTML-based site. These new features are described, along with an overview of the design and implementation.

1 Introduction

Intensional HTML provides a convenient means for creating WWW sites which exist in many versions, without the need to create, and more importantly maintain, each individual version explicitly. The first version, though, was a research prototype, and unsuitable in several respects for general use. Intensional HTML 2 (IHTML 2) provides improved reliability, performance, and usability, as well as some completely new features.

IHTML 2 is an add-on module for the Apache (tm) WWW server [1]. Because it is part of the server, it can process IHTML requests as easily and conveniently as any other type; there is no intrinsic overhead for using it. It provides a variety of new features: selective inclusion of text in a page, based on the requested version, dimensional variables which allow one dimension to use values from another dimension, the ability to create links to pages using different version spaces, and convenient tools for creating intensional WWW sites.

2 IHTML Overview

The need for multi-versioned WWW sites is plain: sites must exist in multiple languages, expertise levels, subscription levels, graphics levels, colours, and so on. When a site varies in more than one of these areas, the number of possible combinations may become very large. IHTML provides an alternative to cloning,
by allowing authors to define large families of pages with relatively few files. The motivation for, and general ideas underlying, intensional HTML are described in [2]; a summary is given in this section.

The objective of IHTML is to allow the creation of sites which exist in many versions, without having to construct and maintain each version independently. The approach is to automate the production of individual versions of each page, from a collection of versioned components. For example, a page containing text and graphics might reasonably vary according to language and graphics level. The non-IHTML solution is to have many versions of the page, one for each combination of language and graphics level. The IHTML solution is to have one page: a framework (not an HTML frame) which includes the appropriate components. The components (text or graphics) will exist in multiple versions, but only those versions for which their content varies. For example, an image might need to exist in two versions: high resolution and low resolution, but would be the same in all languages. A text component might exist in English, French and Turkish versions, but would not need to vary in the graphics dimension. When a version of the page is requested, the main framework page is located, and the appropriate versions of the text and graphics components are included.

There are two key ideas underlying IHTML. First, when a particular version of a page (or component of a page) is requested, the “best fit” version is found. This might be the exact version requested, if it exists. If it does not exist, then the version that is closest to it (in a specific sense, described later) is used instead. This allows the user to provide generic components which correspond with many different versions of a page. Second, links, includes and other HTML features are treated as intensional: they are links to families of versions of a page, rather than individual pages. For example, an anchor such as <A HREF="zork.html"/> is interpreted as a link to the current version of page “zork.html”; the exact version of the page which is reached via this link depends on the version of the current page being browsed.

Some terminology will be helpful here. An extensional (or browsable) page is the page the user browsing the WWW site sees: a particular version of a particular page. An intensional page is the set of all versions of a page. A physical page (or component of a page) is the page (or component) created by the site implementor. In general, an intensional page corresponds with a large number of extensional pages; the many extensional pages are constructed from a much smaller collection of physical components.

Versions and the Best Fit Idea Each IHTML 2 site has a version space in which its pages exist. A version space is a set of dimensions, with a set of indices for each dimension. Each dimension's index set includes a unique vanilla value, which is treated specially when comparing versions. A version is a point in this dimensional space.

Versions are written as a disjunction (using the symbol ‘+’) of dimension specifiers, where each dimension specifier names a dimension and (optionally)
IHTML 2: Design and Implementation

an index in that dimension. The following grammar describes the syntax of versions:

```
<version> ::= <dimspec>
  |  <dimspec> + <version>
<dimspec> ::= <dimname> : [<dimval>]
<dimname> ::= <symbol>
<dimval>  ::= <integer>
  |  <symbol>
```

Symbols are the usual alphanumeric strings; integers are non-negative. If the `<dimval>` is omitted from a `<dimspec>`, it defaults to `vanilla`. Examples of versions are:

```
language:fr+graphics:hir+hire
revision:3+platform:sparc+debugging:2
zork1,zork2+zork3,zork4+zork5,zork6
language:+graphics:
```

and so on.

The rules for comparing dimensions are described fully in [3]; the basic idea and a few examples are given here. Briefly, the algorithm is to find the exact version, if it exists, or to find a unique closest match otherwise. Versions are partially ordered by a refinement relation "\(\subseteq\)”, meaning “refines to” or “is more generic than”, with rules such as:

\[
V \subseteq V + V'
\]

\[
D : N_1 \subseteq D : N_2 \quad \text{if} \quad N_1 < N_2
\]

For example, suppose the requested version were “a:b+c:3”. If that exact version existed, it would be used. If it did not, but “a:b+c:2” did, that would be used. If no version existed which had values for both the “a” and “c” dimensions, but one for “a:b” existed, it would be used. Note that numbers are matched numerically, as one might expect: version \(D : 3\) is a refinement of \(D : 0\), \(D : 1\), and \(D : 2\).

Given the complete set of ordering rules, the best fit is the version which is equal to or more generic than the requested one, and more specific (i.e. closer to the requested version) than all others which are also more generic than the requested one. If there is no unique best fit, it is an error and no match is found. For example, given the requested version above, suppose versions “a:b” and “c:2” exist, but not the requested one. In this case, each one of them is more generic than the requested version, but they are not comparable with each other. It would appear to the user as a standard HTTP 404 “Not Found” error.

Intensional Links and Includes The second key feature of IHTML is that anchors are not links to particular pages, but to families of pages (i.e. to intensional pages, rather than extensional). Which exact page is reached via a link
IHTML 2: Design and Implementation

may depend on the version of the current page as well as the information in the anchor.

For example, consider an example like the one described above, with a top-level page component which exists in a single version, and elements of the page which exist in many versions. Suppose the page is page 5 of a slide show, and contains a link to page 6. The anchor in the top-level component is "<A HREF="page6">". If the user is viewing the "language:french" version of the page, the anchor is interpreted as a link to that version of page 6; if the user is viewing the vanilla version of this page, the link reaches the vanilla version of page 6. The exact same source text creates a link to different places, depending on the version of the current page.

Anchors (as well as includes and other IHTML tags) may include dimension modifiers, making them transversion links. For example, the anchor "<A HREF="page5.html" WMD="lang:english">" is a link to page 5, but with the "lang" dimension set to "english". This kind of link is used to create links to other versions of a page or site, to include other versions of some elements, and so on.

3 Design Overview

The design of IHTML falls naturally into four categories: version representation, site construction, URL-to-filename translation, and IHTML-to-HTML translation. Versions are represented to the site implementor by the strings described in the previous section, with an encoded representation for use in URLs. Versioned objects (HTML files, images, and so on) are directories containing the various physical pages or other components. Site construction is addressed by Unix programs which allow the user to create and manipulate versioned objects without having to deal with encoded versions. URL-to-filename translation is the stage at which the URL for an incoming HTTP "GET" request (possibly containing a version code) is translated to the Unix filename of the best-fit version of the requested page. IHTML-to-HTML translation converts the specified IHTML file to pure HTML, and sends the resulting file to the client.

Version Representation This section covers three topics: how the versions themselves are encoded, how versions are represented in URLs, and how versioned objects are stored on disk.

Version encoding occurs in two steps: conversion to a canonical form, then encoding. The canonical form for versions has not changed from IHTML 1; see [3] for details. The encoding is a simple implementation of arithmetic coding [5], followed by conversion from a bit string to an ASCII string: each successive 6 bits is converted to a character from the range "a-zA-Z0-9_.". The encoding contains all the information present in the original version expression. Here are some examples of encoded versions:
IHTML 2: Design and Implementation

<table>
<thead>
<tr>
<th>Version</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>language:english+graphics:hires M1lw9qwwCBrUThyKZuJZhBmHsDfc</td>
<td>aXHTPmoM8b-JIBLYnRjDkJkGzRjTkq</td>
</tr>
<tr>
<td>zork1:zork2+zork3:zork4</td>
<td>aai</td>
</tr>
<tr>
<td>vanilla</td>
<td></td>
</tr>
</tbody>
</table>

URLs of versioned objects (pages, images, or whatever) are represented as normal URLs, but with a version code inserted between the filename and the suffix of the object. For example, if "aXHTPmoM8b-Jleai!" is the encoding of version "zork1:zork2" (which it is), then the "zork1:zork2" version of the page identified by the URL "http://www.moria.com/rogue.html" is "http://www.moria.com/rogue.aXHTPmoM8b-Jleai.html". Since objects specifiable by URLs have a suffix identifying their type, this seems an appropriate place to put the version code. The IHTML server looks for URLs with version codes as described, and handles them as described below. URLs without such version codes are handled as if by a vanilla Apache server.

Versioned objects such as pages, images, or parts of pages (components) are represented as directories, named as the object itself would be named. For example, the intensional page "zork.html" is represented as a directory named "zork.html". The physical versions are files in the directory; the vanilla version of this page would be "zork.html/aai.html". Two points are important: first, the names of the files contain all information necessary to recover version information; second, the site implementor need never be aware of the directory structure, or the version encodings. This second idea is enabled by the site implementation utilities described in the following paragraphs.

Site Construction Utilities  Clearly, we must not expect users (either site implementors or browsers of intensional sites) to type or understand the encoded versions described in the previous section. Four utilities are provided to hide them from view. They are essentially front ends to standard Unix utilities, providing version encoding and decoding facilities.

icp  This program copies an ordinary file to a specified version of some versioned component or page, or copies a specific version of some component back to an ordinary file.

ivl  Invokes a user-specified editor on a given version of a versioned HTML page or component. The editor specified in the EDITOR environment variable is invoked. If none is specified, v1 is used.

ils  Lists the files in a directory; for versioned things, also lists the versions in which they exist.

irm  Removes a particular version of a component.

URL-to-filename Translation  As described above, URLs in HTTP "GET" requests are of the form:

...URLpath/filename.[vcode.].suff
IHTML 2: Design and Implementation

where the vcode is optional. The server software must translate them to the form:

...Unispath/filename.suff/best-fit-vcode.suff

There are three steps, only the third of which varies from normal URL processing:

1. Prepend the appropriate Unix path. It will be either the server’s “document root”, or a user’s “.www” directory (if the path begins with “userid”).

2. Remove any trailing “path info”.

3. Extract the version code from the URL, find the best fit to this requested version for the named page or other object, and construct the complete filename. If there is no best fit, return an HTTP 404 “Not Found” error.

IHTML-to-HTML Translation

Like server-parsed documents, IHTML pages are translated to pure HTML before being sent to the client. The translation process works as follows:

- Normal text is echoed unchanged.

- HTML tags which are not relevant to IHTML are echoed unchanged.

- HTML tags which include URLs are transformed as follows:
  1. Find the attribute which is to be altered: HREF, SRC, ACTION, or VALUE.
  2. If the tag includes a VERSION attribute, use that as the base version, otherwise use the current version for this page.
  3. If the tag includes a VMOD attribute, merge it into the base version, resulting in the requested version for this tag.
  4. Construct the encoding for this version.
  5. Insert the encoding for this version between the filename and suffix of the URL.
  6. Output the modified tag (not including the VERSION or VMOD attributes).

Tags transformed in this manner are: A, IMG, OPTION, FORM, FRAME.

- IHTML tags are replaced with the appropriate text:
  
  include Process the included file as if it were a normal IHTML request, and copy the result to the output.
  
  echo Echo the current index of the named dimension to the output.
  
  iurl Construct a versioned URL from the HREF attribute, as described above, and copy it to the output.
dumpver  Dump the entire current version to the output (for debugging use, mainly).
starttag  Send "<" to the output.
endtag    Send ">" to the output.

Note: If the HREF attribute is missing from the "A" or "IURL" tags, the current page is used by default.

- Structure tags (ISELECT, ICOLLECT, ICASE) are processed as described in the following section.

4 New Features

User-specified Dimensions  The original version of IHTML required a configuration file specifying the set of acceptable dimensions for pages in a site. This had two negative consequences. Users could not pick which dimensions they wanted; the software itself had to be reconfigured to add a dimension. More importantly, it was impossible to include a link to a different IHTML site, which supported a different set of dimensions. IHTML 2 allows users to use arbitrary (alphanumeric) dimension names.

Anchors, includes, and so on in IHTML 1 specified dimensions as in the following example:

```html
<A HREF="blah" LANGUAGE="blah" BGC="blah" ...>
```

where "LANGUAGE" and "BGC" are dimension names. Given this syntax, the translation software must know which dimensions are possible, so as to distinguish them from arbitrary HTML tag attributes. IHTML 2 uses the following syntax, making it possible to distinguish dimension names without knowing the set of dimensions in advance:

```html
<A HREF="blah" VMOD="LANGUAGE:blah,BGC:blah" ...>
```

Simplified Version Encoding  One important design criterion for IHTML is that the versions in URLs be encoded, so that browsing users cannot request arbitrary versions, but only those allowed by the site implementor. IHTML 1 encoded versions as integers, with a central index file to map integers to human-readable versions. This approach suffers from several limitations:

- It is slow, since the map file must be loaded and parsed on every HTTP request, and possibly modified and saved as well. (Caching the map would entail modifying the server software, which IHTML 1 was intended to avoid.)

- It is dangerous, since loss of the map file results in loss of all version information for the site (physical pages and components are identified by their encoded version integers, not the textual versions).
IHTML 2: Design and Implementation

- It is not clear how to create links to sites that use different version spaces.
- There are synchronization problems, since many instances of the server process could require read/write access to the file at one time.

The one benefit of this approach is that version codes in URLs were short (typically 2 or at most 3 digits).

The IHTML 2 approach is to encode the entire version in filenames, URLs, and anywhere else they are needed. The drawback is that URLs for pages may include long, peculiar strings if the required version is complex. However, there are significant advantages:

- There is no single point of failure (no version file to be corrupted).
- There are no synchronization problems (no version file to get read/write access to).
- It is possible to encode versions for arbitrary version spaces, so one can include links to other versioned sites.
- There is little or no performance penalty.
- The implementation is considerably simpler.

The complexity of filenames and URLs is not a significant problem, since utility programs are provided which prevent the user from having to type the encodings of versions; all the user ever needs to see is the normal textual versions.

**Absolute Version Specifications** Under normal circumstances, an anchor or include is interpreted as being relative to the current version of the page in which the link exists; any dimensions specified in the link augment the current dimension, rather than replacing it. It may, however, occasionally be useful to include a link to a specific version of a page, regardless of the current version of the referring page. One example is a link to the vanilla version of a page. IHTML 2 supports absolute (as opposed to relative) version specifiers, in addition to the usual ones.

For example, the following anchor leads to the vanilla version of the page, regardless of the version of the current page:

```
<a href="zork.html" version="">Back to Vanilla</a>
```

**Selective Inclusion of Text** One common dimension in which a site might vary is background colour. The approach used to implement this effect in IHTML 1 is to include at the top of each file a component which sets the background colour, which (naturally) varies in the “background” dimension. No other components need vary in this dimension. Once the user picks, say, the “background: green” version of the site, this dimension is propagated through links, includes, etc. in the usual way. The “background” component itself is
composed of several small files, one per index (i.e. colour) in the background dimension.

It turns out that this general idea occurs frequently in IHTML sites. It works, but it is somewhat tedious to create the many small (frequently 1 line) files that are necessary. IHTML 2 provides a *selective inclusion* feature very much like a *switch* statement in the C programming language. Its syntax is as follows:

```
<ISELECT>
  <ICASE VERSION=\text{v1}...text1...
  <ICASE VERSION=\text{v2}...text2...
  <ICASE VERSION=\text{v3}...text3...
  ...
</ISELECT>
```

In general, the output includes the `text_i` corresponding with the first `\text{v_i}` which is equal to or less specific than the current version. Note that the trailing `</ICASE>` tags are optional. If they are omitted, the output includes the following case(s) until a `</ICASE>` is seen, or the closing `</ISELECT>` is seen. This is analogous to the fall-through behaviour of a case without a closing break in a C *switch* statement. An `</ICASE>` with no version specified is considered vanilla, and refines to any version.

Consider the "background" example. Using this approach, the "background" component, instead of consisting of many small files, could be a single (vanilla) file containing the following ISELECT:

```
<ISELECT>
  <ICASE VERSION="background:blue"><BODY BGCOLOR="blue"></ICASE>
  <ICASE VERSION="background:red"><BODY BGCOLOR="red"></ICASE>
  <ICASE VERSION="background:green"><BODY BGCOLOR="green"></ICASE>
  <ICASE VERSION="background:zork"><BODY BGCOLOR="zork"></ICASE>
  <ICASE><BODY BGCOLOR="black"></ICASE>
</ISELECT>
```

This idiom turns out to be extremely useful.

A similar structure called *ICollect*, with similar syntax (just substitute "COLLECT" for "SELECT") includes *every* case which matches the current version.

**Dimension Variables** It is sometimes useful to use the current index of one dimension as the index of another, or possibly to use the current index of one as the dimension for another dimension specifier. For example, suppose one wanted the ability to set the "visited link colour" of a page to the same as the current "unvisited link colour". A link such as the following would do the job, assuming the two dimensions were named "vlc" and "ulc", and all the necessary includes were defined:

```
<A VMOD="vlc:$ulc">set link colour</A>
```
IHTML 2: Design and Implementation

This actually introduces two features: the use of $ to refer to the current index of a dimension, and the use of default URLs: if you omit the HREF attribute from an anchor, it defaults to the current page. This behaviour makes it very simple to create a component with anchors which modify the current page in a variety of ways, no matter what page the file is included in.

It is also possible to use dimension variables to specify dimensions:

    <A VMOD="$zork:rogue">do something...</A>

The current index of dimension zork will be treated as a dimension and the link will set that dimension to index rogue.

Site Construction Utilities Perhaps the single biggest weakness of IHTML 1 is the difficulty of creating intensional sites. Users are required to understand how versions are encoded, name files carefully to reflect the version encodings in the previously described map file, and so on. IHTML 2 addresses this difficulty by providing utility programs which insulate the user from implementation details of IHTML. The objective is to allow the user to view an intensional page as a single entity, which happens to exist in multiple versions.

Details on these programs are found in the Design Overview and Implementation Details sections.

5 Implementation Details

The implementation of this software is mostly straightforward. A few points of potential interest are discussed here.

Identifying Versioned Objects One of the things which the software must do in several instances is decide whether a name refers to a versioned thing (in which case it should be searched for versions, etc), or whether it is just a normal file, in which case versioning is inappropriate. If all of the following criteria are true, the object is considered to be an IHTML versioned thing, and is treated accordingly:

- It is a directory.
- Its name is of the form filename.suffix.
- It contains at least one file of the form vcode.suffix, where vcode decodes into a legal IHTML version.

The directory may contain other files as well; they will be ignored.

The Apache Module Interface IHTML is implemented as an Apache module, using the provided API (application programming interface). Modules can insert hooks into the request serving process at a variety of points:
IHTML 2: Design and Implementation

- URL-to-filename translation
- header parsing
- access checking (based on client IP address, HTTP user authentication, and file/directory permission checking)
- MIME type checking or setting
- document serving (by MIME type)
- generating log records

The IHTML module has hooks at two points: URL-to-filename translation and document serving. All incoming "GET" requests are checked to see if they refer to versioned objects. If they do, they are processed as described in this paper. Otherwise, they are passed on to the standard Apache software for normal handling. If the request was handled as an IHTML request, and the MIME type is "text/html", then the file is filtered through the IHTML-to-HTML translator when it is sent to the client (at document-serving time).

Code Re-use A considerable amount of code is shared between the Apache IHTML module and the Unix site construction utility programs, especially for version encoding and decoding. Since Apache modules must follow very strict guidelines for memory allocation and I/O, a small library of stubs was created, simulating Apache's allocation and I/O routines. In this way, when the code is compiled into the Apache server, it behaves as an Apache module should; when it is compiled into the Unix utilities, it does normal Unix-style allocation and I/O. This stub approach avoids the evils of #ifdef, the usual approach to "portable" programming in the Unix environment.

6 Future Work

IHTML is still very much a work-in-progress. This section lists a few things which ought to be explored.

Version Manipulation The introduction of dimension variables could be the beginning of a more sophisticated treatment of versions. It seems that it might be useful for IHTML pages to manipulate versions in a variety of ways: examine them, merge them, perform tests on them, construct them from scratch, and so on. Some of these things are already present, in limited forms, but it would be interesting to explore some more general ideas of a version manipulation language.
JavaScript and IHTML  The possibilities suggested in the previous paragraph open up the further question of interactions between JavaScript and IHTML. At the moment, interaction is extremely limited. One useful capability which is currently impossible is to be able to select values for several dimensions in an HTML form, then go to the version thus described with a single HTTP transaction.

Enhanced Version Space  The version space provided by IHTML 2 is actually more limited than that of IHTML 1; it should certainly be extended until it is at least as powerful. There are also new areas to explore: a “greatest lower bound” operator to complement the “+” (least upper bound), and an index value more refined than vanilla, but less refined than any other value (to indicate that a dimension must have some value other than vanilla, but it doesn’t matter what).

Applets  The current approach depends on the ability to name files in an arbitrary way. The Java class loader, however, is sensitive to filenames. Two problems arise. Suppose the class loader is told to ask for version “a:b+c:d” of a file, but the best fit is version “a:b”. The server will send along version “a:b”. However, the class loader will also expect the class in the file to be named with the code for version “a:b+c:d”, and will complain bitterly if it is not. The second problem arises once we solve the first one (there are a number of ugly solutions to it). It is necessary that any classes used by the main applet class also be the appropriate version, but we have no way to tell the class loader what to ask for. One solution is to process Java class files somewhat the way IHTML files are processed, translating class names to the ones the class loader is expecting. At first, this seemed quite ugly, but the alternative is modifying the class loader, which is worse (since it is a browser-side modification).

Aggregates  IHTML 1 supports a notion of aggregates. Instead of finding the best fit to a requested version, it finds all matches. This facility is not present in IHTML 2, because (in the opinion of the author) aggregates require more thought to be fully understood. There are a few issues: Should we find all best fits, or all that refine to the requested version? How should the collected items be ordered? How should they be formatted? How should the site implementor specify which dimensions to aggregate on, and when to do it? We have partial answers to these questions, but more thought is required to get a clear picture.

Non-HTML applications  There is nothing especially HTML-specific about a lot of this work. It could apply to other versioned text, such as software or non-HTML documents. In fact, many of the original ideas for this work arose from a software versioning system called Lemur [6]; perhaps some of the ideas developed here could usefully be transferred back to versioned software development.
IHTML 2: Design and Implementation

IHTML editor Although the Unix utilities provided with IHTML 2 make the site implementation job easier, there is considerable room for improvement. With the trend towards the use of HTML editors rather than hand-coding of HTML, we must consider the possibility of an IHTML editor. The interesting question is: how should we present a versioned entity (such as a WWW page) to the user of a WYSIWFY editor?

7 Summary

IHTML 2 provides a variety of new features to IHTML, as well as improved usability, performance, and reliability. Key new features include the document structuring capabilities of ISELECT and ICOLLECT, dimension variables, a more flexible version coding system, and utilities to make site construction simpler.

There remains, however, a great deal of work to be done to make this a viable tool for general use. In particular, some kind of WYSIWFY IHTML editor is probably a prerequisite for use outside the programming community. Support for applets is also essential, or close to it.

Despite these limitations, though, IHTML 2 is a significant step forward from IHTML 1, in overall usefulness. It has been used by a variety of users, with reasonable success, suggesting that some critical threshold of usefulness has been crossed, or can be without too much more effort.

References


