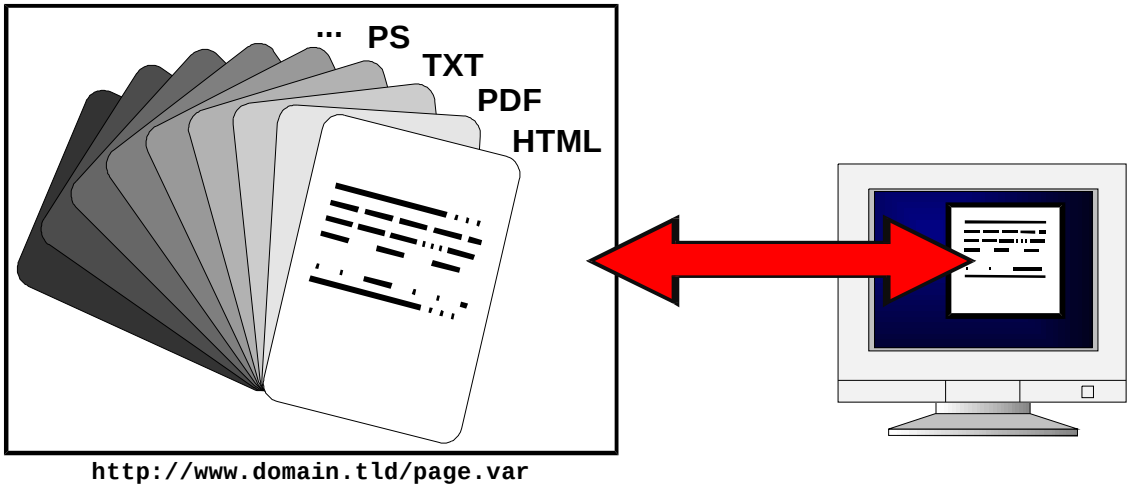
Transparent Content Negotiation

Lars Eilebrecht
<lars@apache.org>

1 Content Negotiation

If more than one version of the same resource, a GIF, JPEG, and PNG file of an image, for example, is available on the server, Apache is able to select one of these files based on the client's preferences and capabilities. This mechanism is called *Content Negotiation*, or more accurately *Content Selection*.

The prerequisite is that different variants of a specific resource, a HTML document, an image, etc., have been made available on the server. For example, a web client with a preference for PDF over Postscript documents would receive the PDF instead of the Postscript variant.



In order to use content negotiation the module `mod_negotiation` has to be added to the Apache server. The module includes two different implementations of content negotiation.

The normal way how Apache decides about the best variant is to read the HTTP headers submitted by the web client. This means that the selection of a variant is solely done on the server side. This is sometimes referred to as *HTTP/1.0 Content Negotiation* or *Apache Content Negotiation* (ACN). Since Apache version 1.3.4, client−side content negotiation is available, too. In this case Apache doesn't send a variant to the client, but responds with a list of all available variants for that resource, to allow the client itself to select a variant that best matches the client's capabilities. For the user the whole process of selection is usually completely transparent, which is why this mechanism is often referred to as *Transparent Content Negotiation* (TCN). Apache complies to the specification of Transparent Content Negotiation in RFC 2295[1]. Indeed it may happen that a web client, dependent on the implementation of TCN, presents the user a list of all variants allowing him to select one manually. Even if a web client supports TCN he usually leaves the selection of a variant to the web server. Regarding this, Apache supports the *Remote Variant Selection Algorithm 1.0* (RVSA/1.0) described in RFC 2296.

1 RFC 2295 defines "Feature Negotiation" which is not yet supported by Apache 1.3.

1.1 Standard Apache Content Negotiation

All preferences and capabilities of a web client are submitted to the server via a number of different HTTP headers. These headers are `Accept`, `Accept−Charset`, `Accept−Encoding` and `Accept−Language`. These headers result in four different possibilities for Apache to selecting the *best* variant on behalf of a client:

- MIME type
- Character set
- Encoding
- Language

There are two different mechanisms supported by Apache for resolving what to send a client. The most flexible one is the use of *Type−Maps* allowing explicit definitions of a resource's variants. Less flexible, but simpler and more convenient to use is the *MultiViews* feature.

1.1.1 Apache Negotiation Algorithm

In order to determine which one of a selection of documents or images should be returned to a client a special negotiation algorithm is used. The following paragraphs are taken out of the Apache documentation[2] explaining the different negotiation steps of the *Apache Negotiation Algorithm*:

1. First, for each dimension of the negotiation, check the appropriate `Accept*` header field and assign a quality to each variant. If the `Accept*` header for any dimension implies that this variant is not acceptable, eliminate it. If no variants remain, go to step 4.
2. Select the *best* variant by a process of elimination. Each of the following tests is applied in order. Any variants not selected at each test are eliminated. After each test, if only one variant remains, select it as the best match and proceed to step 3. If more than one variant remains, move on to the next test.
    1. Multiply the quality factor from the `Accept` header with the quality−of−source factor for this variant's media type, and select the variants with the highest value.
    2. Select the variants with the highest language quality factor.
    3. Select the variants with the best language match, using either the order of languages in the `Accept−Language` header (if present), or else else the order of languages in the `LanguagePriority` directive (if present).
    4. Select the variants with the highest *level* media parameter (used to give the version of `text/html` media types).
    5. Select variants with the best character set media parameters, as given on the Accept−Charset header line. Charset ISO−8859−1 is acceptable unless explicitly excluded. Variants with a `text/*` media type but not explicitly associated with a particular charset are assumed to be in ISO−8859−1.

2 `http://www.apache.org/docs/manual/content−negotiation.html`

6. Select those variants which have associated charset media parameters that are not ISO−8859−1. If there are no such variants, select all variants instead.
7. Select the variants with the best encoding. If there are variants with an encoding that is acceptable to the user−agent, select only these variants. Otherwise if there is a mix of encoded and non−encoded variants, select only the unencoded variants. If either all variants are encoded or all variants are not encoded, select all variants.
8. Select the variants with the smallest content length.
9. Select the first variant of those remaining. This will be either the first listed in the type−map file, or when variants are read from the directory, the one whose file name comes first when sorted using ASCII code order.

3. The algorithm has now selected one *best* variant, so return it as the response. The HTTP response header Vary is set to indicate the dimensions of negotiation (browsers and caches can use this information when caching the resource). End.
4. To get here means no variant was selected (because none are acceptable to the browser). Return a 406 status (meaning *No acceptable representation*) with a response body consisting of an HTML document listing the available variants. Also set the HTTP `Vary` header to indicate the dimensions of variance.

Fiddling with Quality Values

Apache sometimes changes the quality values from what would be expected by a strict interpretation of the Apache negotiation algorithm above. This is to get a better result from the algorithm for browsers which do not send full or accurate information. Some of the most popular browsers send `Accept` header information which would otherwise result in the selection of the wrong variant in many cases. If a browser sends full and correct information these fiddles will not be applied.

Media Types and Wildcards

The `Accept` request header indicates preferences for media types. It can also include *wildcard* media types, such as "`image/*`" or "`*/*`" where the `*` matches any string. So a request including:

```
Accept: image/*, */*
```

would indicate that any type starting "`image/`" is acceptable, as is any other type (so the first "`image/*`" is redundant).

Some browsers routinely send wildcards in addition to explicit types they can handle. For example:

```
Accept: text/html, text/plain, image/gif, image/jpeg, */*
```

The intention of this is to indicate that the explicitly listed types are preferred, but if a different representation is available, that is ok too. However under the basic algorithm, as given above, the `*/*` wildcard has exactly equal preference to all the other types, so they are not being preferred. The browser should really have sent a request with a lower quality (preference) value for `*.*`, such as:

```
Accept: text/html, text/plain, image/gif, image/jpeg, */*; q=0.01
```

The explicit types have no quality factor, so they default to a preference of 1.0 (the highest). The wildcard `*/*` is given a low preference of 0.01, so other types will only be returned if no variant matches an explicitly listed type.

If the `Accept` header contains no q factors at all, Apache sets the q value of "`*/*`", if present, to 0.01 to emulate the desired behavior. It also sets the q value of wildcards of the format "`type/*`" to 0.02 (so these are preferred over matches against "`*/*`". If any media type on the `Accept` header contains a q factor, these special values are not applied, so requests from browsers which send the correct information to start with work as expected.

(See also the explanation of the *Remote Variant Selection Algorithm* at the end of this document.)

1.1.2 Type Maps

A *Type Map* is a simple text file containing definitions of all available variants for a specific resource. Type maps are usually identified by the file extension "`.var`" as an abbreviation for *variant*.

Activating Type Maps

Type maps are activated by assigning an Apache handler called "`type-map`" to a file extension. As already mentioned this is usually the extension "`.var`".

```
AddHandler type-map var
```

If you use the `AddHandler` directive globally in your server configuration, files with a "`.var`" extension in all server directories are defined as a type map. If you wish to limit the use of type maps to a specific part of your web server this is done by using a `Location` or `Directory` directive, e.g.:

```
<Location /test>
AddHandler type-map var
</Location>
```

Using Type-Maps

As mentioned before there are four different dimensions of content negotiation: Media type (`Accept` header), language (`Accept-Language` header), character set (`Accept-Charset` header) and content encoding (`Accept-Encoding` header). For every variant it is possible to make definitions for one or more dimensions of negotiation in a type map.

A type map has the same format like an electronic mail header (see RFC 822). The headers allowed are:

URI

A URI (Uniform Resource Identifier) entry is used to set the file name of a variant, e.g.:

```
URI: page.html
```

or

```
URI: ../page.html
```

The file name has to be defined relative to the current directory. It is not possible to use an absolute path.

Content−Type

The media type and quality score (qs) of a variant is defined by a Content−Type entry. Both values have to be separated by a semicolon. As an additional parameter (Charset) the character set of the content can be defined.

The definition of a Content−Type entry is mandatory for every variant, even if all variants in a type map have the same media type.

Examples:

```
Content−Type: text/html
Content−Type: image/gif; qs=0.5
Content−Type: text/plain; qs=0.4; charset=iso−8859−5
```

Content−Length

This entry defines the file size (bytes) of a variant. Apache checks for the file size itself if this entry is missing, therefore it is save to omit all Content−Length entries.

Content−Language

The language used for a variant is defined via Content−Language. Allowed values are double−digit abbreviations of the corresponding language as defined in ISO 639 and RFC 1766, e.g., "en" for English, "de" for German or "fr" for French.

Example:

```
Content−Language: de
```

Since there can be more than just one variant for one language, e.g., American English and British English, an additional extension, separated by a dash, may be added.

Examples:

```
Content−Language: en−US
Content−Language: en−GB
```

If a client requests the language "en−GB" the server replies with the "en−GB" variant, if available. If not, he looks for other "en" variants. Accordingly a request for "en" would yield a "en−GB" variant if no other "en" variant is available.

Content−Encoding

If a file has been encoded with compress or gzip this can be defined with Content−Encoding.

Examples:

```
Content−Encoding: gzip
Content−Encoding: compress
```

Description

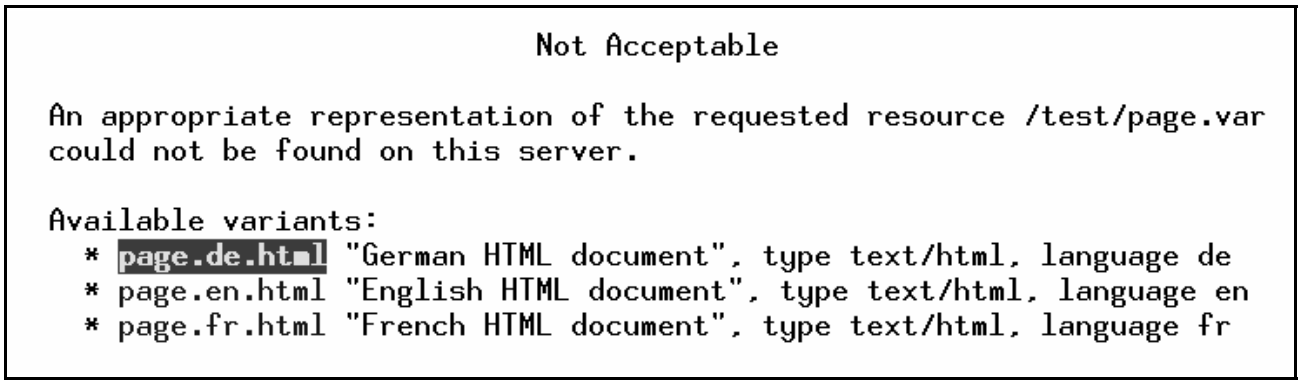
As the name already denotes, it contains a description of the corresponding variant. A Description entry is optional, but if there are descriptions in a type map they are used by

Apache if no matching variant could be found for a client request. In that case an error message (status code 406, Not Acceptable) is returned to the client including a list of all available variants together with their descriptions. This allows a user to manually select a variant.

If a description contains a white space it must be quoted.

Examples:

```
Description: "Original english version"
Description: "German HTML document"
Description: Postscript
```



Every entry mentioned above may be used multiple times in a type map, but only once for a single variant. For every variant at least an URI and Content−Type entry must be defined in a type map. Variant entries in a type map are separated by an empty line. Lines starting with a "#" are treated as comments and therefore ignored by Apache.

Example of a Type−Map file

```
#
# type-map: /test/page.var
#

URI: page.de.html
Content-Type: text/html; qs=0.8
Content-Language: de
Description: "German HTML document"

URI: page.en.html
Content-Type: text/html; qs=0.8
Content-Language: en
Description: "English HTML document"

URI: page.txt
Content-Type: text/plain; qs=0.1
Content-Language: en
Description: "English plain text document"

URI: page.pdf
Content-Type: application/pdf; qs=1
Content-Language: en
Description: "English PDF document"
```

A client requesting the URL `http://www.domain.tld/test/page.var` would receive one of the variants `page.de.html`, `page.en.html`, `page.txt` or `page.pdf`. If the client prefers only german documents he would receive `page.de.html` and not the PDF document which has a higher quality score, but is not available in German. Only if the client accepts English documents the file `page.pdf` would be returned to the client.

1.1.3 Dimensions of Negotiation

Type Negotiation

A client uses the Accept header to submit a list of acceptable media types to the server, e.g., `text/html`, `image/png` or `application/pdf`. In addition a quality value "q" can be specified for every media type. This value is a floating−point number in the range 0.0 to 1.0, indicating the relative quality of this media type to the other media types. The default is 1.0 if no quality value is specified.

Example header:

```
Accept: image/png;q=1, image/gif;q=0.5, image/jpeg;q=0.7
```

This header expresses the client's capability to handle PNG, GIF and JPG images and shows a preference of PNG images (`q=1`) over all other image types. If there is no PNG variant Apache responds with a JPG image and if there is even no JPG image he will look for a GIF image. If there is no acceptable variant Apache responds with an 406 error message (see example above).

Example of a type map for three image files:

```
#
# type−map: picture.var
#

URI: picture.png
Content−Type: image/png; qs=0.6
Description: "Truecolor PNG image"

URI: picture.gif
Content−Type: image/gif; qs=1
Description: "256color GIF image"

URI: picture.jpg
Content−Type: image/jpeg; qs=0.6
Description: "Truecolor JPEG image with 70% quality−level"
```

A request for `picture.var` with the `Accept` header given above would result in the PNG variant being returned to the web client. The quality values of the client are combined with the quality scores defined on the server−side. The variant with the highest resulting value is the *best* variant and therefore returned to the client.

If the client changes the quality value for the media type `image/png` from 1 to 0.5, the result would be the GIF variant of the image.

Language Negotiation

If a document is available in multiple languages, e.g., German and English, the `Accept-Language` header can be used to select the best matching variant.

Example header:

```
Accept-Language: de, en:q=0.9, fr:q=0.2
```

Expressed verbally, this header means: "I'm preferring German, but also English if there is no German document, and if all else fails I'll take French documents as well".

Example of a type map for a HTML document available in German, English and French:

```
#
# type-map: page.var
#

URI: page.html.de
Content-Type: text/html
Content-Language: de
Description: "German document (original version)"

URI: page.html.en
Content-Type: text/html
Content-Language: en
Description: "English document (translated version)"

URI: page.html.fr
Content-Type: text/html
Content-Language: fr
Description: "French document (translated version)"
```

A request for `page.var` with the given `Accept-Language` header would yield the German variant of the HTML document.

Charset Negotiation

A list of supported character sets can be submitted by the web client to the server via the `Accept-Charset` header. The default character set for HTML documents is ISO-8859-1.

Example header:

```
Accept-Charset: iso-8859-1, unicode-1;q=0.8
```

Example type map for two HTML documents:

```
#
# type-map: page.var
#

URI: page.html
Content-Type: text/html; qs=1; charset=iso-8859-1
Description: "test document (iso)"

URI: page.uni.html
Content-Type: text/html; qs=1; charset=unicode-1-1
Description: "test document (unicode)"
```

Encoding Negotiation

The `Accept-Encoding` header is used by a web client to signal the server which ways of encoding he supports.

Example header:

```
Accept-Encoding: gzip, compress
```

Example type map for a postscript document:

```
#
# type-map: info.var
#

URI: info.ps.Z
Content-Type: application/postscript; qs=0.8
Content-Encoding: compress
Description: "info document (compress)"

URI: info.ps.gz
Content-Type: application/postscript; qs=1
Content-Encoding: gzip
Description: "info document (gzip)"
```

Whether a client supports `compress` or `gzip` Apache either replies with `info.ps.Z` or with `info.ps.gz`.

Combination of different negotiation dimensions

As shown in a previous example it is possible to combine two or more dimensions of negotiation in a single type map.

The following example shows a type map for a resource with HTML and plain text variants, both available in two different character sets and in German and English, respectively:

```
#
# type-map: extreme.var
#

URI: extreme-iso.html.de
Content-Type: text/html; qs=0.9; charset=iso-8859-1
Content-Language: de
Description: "German HTML document, iso-8859-1"

URI: extreme-iso.html.en
Content-Type: text/html; qs=0.9; charset=iso-8859-1
Content-Language: en
Description: "English HTML document, iso-8859-1"

URI: extreme-iso.txt.de
Content-Type: text/plain; qs=0.2; charset=iso-8859-1
Content-Language: de
Description: "German plain text document, iso-8859-1"

URI: extreme-iso.txt.en
Content-Type: text/html; qs=0.2; charset=iso-8859-1
Content-Language: en
```

```
Description: "English plain text document, iso-8859-1"

URI: extreme-uni.html.de
Content-Type: text/html; qs=0.9; charset=unicode-1-1
Content-Language: de
Description: "German HTML document, unicode-1-1"

URI: extreme-uni.html.en
Content-Type: text/html; qs=0.9; charset=unicode-1-1
Content-Language: en
Description: "English HTML document, unicode-1-1"

URI: extreme-uni.txt.de
Content-Type: text/plain; qs=0.2; charset=unicode-1-1
Content-Language: de
Description: "German plain text document, unicode-1-1"

URI: extreme-uni.txt.en
Content-Type: text/plain; qs=0.2; charset=unicode-1-1
Content-Language: en
Description: "English plain text document, unicode-1-1"
```

1.1.4 MultiViews

As you can see by looking at the previous example, the creation and maintenance of type maps is quite time consuming. Apache usually looks at the file extension to find out the media type and encoding of a resource. Doing the same for all variants of a resource would allow Apache to create a type map itself.

This is exactly the mechanism which is used for the *MultiViews* search feature. The use of MultiViews is less flexible than a type map, but a lot simpler to configure.

MultiViews are activated in the server configuration via the option `MultiViews` of the `Options` directive which is usually defined inside a `Location` or `Directory` section, e.g.:

```
<Location /test>
Options +MultiViews
</Location>
```

Please note that `Options All` does not activate MultiViews, it has to be specified additionally.

The functionality of a MultiViews search is quite simple. If the server receives a request for a non existing file, e.g., `/test/page`, and the MultiViews feature is enabled for the directory, Apache reads the directory looking for all files starting with "`page`". If he finds files like `page.html`, `page.txt`, etc. he effectively fakes up a type map which names all those files, assigning them the same media types and content encodings it would have if the client had asked for one of them by name.

If the MultiViews search yields a type map, this map is used for negotiation. That way it is possible to combine a MultiViews search with type maps. The use of type maps could be limited to special cases, for example.

Example: A directory `/test` contains the three files `page.html`, `page.txt` and `page.pdf`. A client requests `http://www.domain.tld/test/page` and submits the following `Accept` header:

```
Accept: text/html;q=1, text/plain;q=0.5, application/pdf;q=0.8
```

The server replies with the file `page.html`, due to the higher quality value. If the client doesn't specify any quality values the result is not predictable. This is a drawback of using MultiViews instead of type maps, because it is not possible to set any quality scores on the server−side.

File name extensions and MultiViews

MultiViews are very convenient to use for language negotiation. As mentioned before the language of a document is denoted by an additional file name extension according to the specification in ISO 639 and RFC 1766.

A file name usually has the following syntax:

`filename.media−type.language.charset.encoding`

Apart from the file name itself, the ordering of all other extensions can be changed if necessary.

Examples:

```
page.html.de
page.de.html
page.html.ja.jis
page.html.jis.ja
info.text.de.gz
info.gz.txt.de
```

In order for Apache to know that "`ja`" means Japanese, for example, the extension has to be defined in the server configuration. This is done with the `AddLanguage` directive, e.g.:

```
AddLanguage de .de
AddLanguage en .en
AddLanguage pl .po
```

If a web client does not submit an `Accept−Language` header the server selects a variant according to the setting of the `LanguagePriority` directive. This directive sets the priority for different language variants on the server side.

In the following example "`en`" has the highest and "`pl`" the lowest priority:

```
LanguagePriority en de pl
```

If a file has no language extension the server is not able to determine the language of it. Via the `DefaultLanguage` directive it is possible to define a default language for variants according to requirements. The directive may be used globally in the server configuration, inside a `VirtualHost`, `Directory`, `Location` or `Files` section or inside a `.htaccess` file. This is very useful if you are migrating from a single to a multiple language web site and don't want to change all your old file names.

Example:

```
<LocationMatch \.html$>
DefaultLanguage de
</LocationMatch>
```

With this configuration all HTML files without a language extension are treated as being a German HTML document. The "`de`" language extension may be used as well.

Another directive which is very useful when using MultiViews is `AddCharset`. It allows for defining a different character set for file names with a specific extension. This may be necessary if the default character set ISO−8859−1 can not or should not be used.

An example of such a case are Chinese documents with different character sets:

```
AddCharset BIG5 big5
AddCharset EUC−TW euctw
AddCharset GB2312 gb
AddLanguage zh .zh
```

This configuration defines the three possible character sets BIG5, EUC−TW, GB2312, and via `AddLanguage` the language Chinese (`zh`) itself. If the content of a document is written with the BIG5 character set, both extensions "`big5`" and "`zh`" must be added to the variant's file name.

Example:

```
page.html.big5.zh
page.html.zh.big5
```

Apart from `AddCharset` the default character set can be changed by using `AddDefaultCharset`, e.g.:

```
AddDefaultCharset ISO−2022−JP
```

The same applies to all other unknown extensions: They have to be defined in the server configuration. Apart from the directive mentioned above there is `AddEncoding` to define encoding extensions and `AddType` to define additional media types.

Referencing MultiViews resources

It is possible to reference MultiViews resources in a hyperlink in different ways, dependent on the ordering of its file name extensions (language, encoding, etc.).

Example:

```
page.html.ja.jis
```

This document may be referenced with one of the following names:

```
page
page.html
page.html.ja
page.html.ja.jis
```

If the variant has the file name

```
page.ja.html.jis
```

the following names may be used:

```
page
page.ja
page.ja.html
page.ja.html.jis
```

As you can see it is possible to omit extensions starting from the right to the left, but it is not possible to omit an other extension or to change the ordering.

Thus using the following names would result in a *Not found* error message:

```
page.ja.jis
page.jis.ja
page.html
```

Special treatment of CGI scripts

If a MultiViews search finds a CGI script and the request was a POST, or a GET with QUERY_ARGS or PATH_INFO, the CGI script is given an extremely high quality score resulting in the invocation of the script. For other request types a low quality score is given to the script which generally causes one of the other variants (if any) to be retrieved.

1.2 Transparent Content Negotiation

As already mentioned in the beginning of this chapter Apache supports *Transparent Content Negotiation* (TCN) since version 1.3.4. From a configuration point of view there are no differences compared to normal *Apache Content Negotiation* (ACN).

All TCN features are active if type maps are in use or the MultiViews search has been enabled, but the use TCN is limited to web clients signaling support for TCN. This is done by a specific HTTP header named `Negotiate`.

If a web client submits the header:

```
Negotiate: trans
```

he signals the server that he supports TCN and wishes to receive a list of all available variants on every response from the server.

If there are three variants on the server, for example:

```
page.html.d
page.html.ja.jis
page.ps.en
```

A request for the document `page` would yield the following response:

```
HTTP/1.1 300 Multiple Choices
Date: Wed, 27 Sep 2000 23:49:31 GMT
Server: Apache/1.3.12 (Unix)
Alternates: {"page.html.de" 1 {type text/html}
             {language de} {length 1877}},
            {"page.html.ja.jis" 1 {type text/html}
             {charset iso-2022-jp} {language ja} {length 1623}},
            {"page.ps.de" 1 {type application/postscript}
             {language de} {length 2326}}
Vary: negotiate,accept,accept-language,accept-charset
TCN: list
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>300 Multiple Choices</TITLE>
</HEAD><BODY>
<H1>Multiple Choices</H1>
Available variants:
<ul>
```

```
<li><a href="page.html.de">page.html.de</a> ,
    type text/html, language de
<li><a href="page.html.ja.jis">page.html.ja.jis</a> ,
    type text/html, language ja, charset iso-2022-jp
<li><a href="page.ps.de">page.ps.de</a> ,
    type application/postscript, language de
</ul>
<HR>
<ADDRESS>Apache/1.3.12 Server at
         www.domain.tld Port 80</ADDRESS>
</BODY></HTML>
```

The most important HTTP headers related to TCN are `Alternates` (variant list), `Vary` (applied dimensions of variance), and `TCN` (response type). Based on these headers the web client is able to transparently select a variant best matching its preferences and capabilities and request this variant from the server.

If there is no matching variant the web client is not able to transparently select one and instead displays the variant list to the user for manual selection. This list is identical to the ACN selection list where a server couldn't find a matching variant, but with ACN this response has the status code 406 (*Not acceptable*) and with TCN it has the status code 300 (*Multiple Choices*).

1.2.1 Remote Variant Selection Algorithm

In most cases a client supporting TCN would use the following HTTP header in its requests:

```
Negotiate: 1.0
```

This denotes that the server (if TCN is supported) should use the *Remote Variant Selection Algorithm 1.0* (RVSA/1.0) to handle negotiation on the server-side. Even if the server selects a variant on behalf of the client the HTTP headers `Alternates` and `Vary` are always returned to the web client. This allows the client, if necessary, to select another variant and request it from the server.

The RVSA is very similar to Apache's negotiation algorithm. Both algorithms use the client's `Accept` headers to select the *best* variant. Indeed there are a few unusual characteristics of Apache's negotiation algorithm, like workarounds for bugs of some commonly used web browsers and special treatment of `Accept` headers without any quality values. Please see the explanation of the *Apache Negotiation Algorithm* for more details.