Implementing Content Negotiation using CC/PP and WAP UAProf

Mark H. Butler
Information Infrastructure Laboratory
HP Laboratories Bristol
HPL-2001-190
August 7th , 2001*

E-mail: marbut@hplb.hpl.hp.com

device independence, content negotiation, composite capabilities / preferences profile, CC/PP, resource description framework, RDF, jena, WAP, wireless access protocol forum, user agent profile, UAProf

Content negotiation is a technique relevant to device independence that allows servers to provide clients with the most appropriate resource from a number of alternates. Several standards have been proposed for content negotiation including HTTP/1.1 server based content negotiation, media feature sets and most recently the Composite Capabilities / Preferences Profile (CC/PP). CC/PP, unlike the other two methods, is only concerned with the client profile and does not specify mechanisms for describing alternate versions of content or matching client profiles to content descriptions. In order to better understand how CC/PP may be used this report describes an implementation of HTTP/1.1-style content negotiation that uses CC/PP client profiles and RDF content descriptions. The Jena RDF Framework developed at HP Labs is used to implement a negotiation algorithm similar to that used by Apache Web Server. As CC/PP is compatible with the forthcoming Wireless Access Protocol (WAP) User Agent Profile (UAProf) these techniques are applicable to the next generation of WAP devices. This is demonstrated using an example profile taken from the current WAP Forum documentation.

* Internal Accession Date Only Approved for External Publication
© Copyright Hewlett-Packard Company 2001

1 Introduction

Content negotiation allows servers to provide a client with the most appropriate resource from a number of alternates and is a useful technique for device independence. Specifically the chief difference between device independence[1] and web accessibility is that accessibility guidelines[2] require that all non-textual content has a text alternate whereas device independence may require multiple alternates. For example we might require different sized versions of the same image in different formats for a PC and WAP phone.

Several standards have been proposed for content negotiation including HTTP/1.1 server based content negotiation[3], media feature sets[4], user agent string detection[5] and Composite Capabilities / Preferences Profile (CC/PP)[6]. CC/PP, unlike server based content negotiation or media feature sets, is only concerned with the client profile and does not specify mechanisms for describing alternate versions of content or matching client profiles to content descriptions. CC/PP is designed to be compatible with the Wireless Access Protocol (WAP) Forum[7] standard User Agent Profile (UAProf). This defines the device profile that will be used in the second generation of WAP devices.

In order to better understand how CC/PP may be used this report describes an implementation of HTTP/1.1-style content negotiation that uses CC/PP client profiles. As CC/PP is based on the W3C Resource Description Framework (RDF)[8] it was decided to use RDF to describe alternate versions of content. The Jena RDF Framework[9] developed by Brian McBride at HP Labs Bristol has been used to implement a negotiation algorithm derived from the one in the Apache Web Server[10]. The implementation can be configured to use different CC/PP vocabularies via an external XML file so that it can process UAProf profiles. This is demonstrated using an example profile from the current WAP Forum Working Draft specification on UAProf[11]. The implementation of the negotiation algorithm, written in Java, is available from the external HP Labs website[12] under an open source license[13].

2 An overview of HTTP/1.1 Content Negotiation

The HTTP/1.1 specification[3] describes server based content negotiation. This particular type of content negotiation tries to match resources to a client based on collections of attributes. For example a resource in the JPEG file format is only sent to the client if it is capable of displaying JPEGs. This matching process requires the attribute of the client to exactly match the corresponding attribute on the resource or a wildcard attribute of the client to match the corresponding attribute on the resource.

In addition, it is possible to specify preferences for attributes of a specific value in the client profile as well as preferences for specific variants of a resource i.e. that a client prefers French documents to English documents or a GIF image resource is preferred over an ASCII art image resource. The client profile can also specify the maximum size of a resource with a specific attribute that the client is willing to accept. For example this means that a client can specify it will not accept JPEG files over a certain maximum size. Where multiple resources are acceptable to a client, the negotiation algorithm multiplies the preference values together in order to determine an overall preference for each resource and then picks the resource with the highest value. Subsequent sections will describe the format of these client and resource

profiles along with a more formal description of the negotiation algorithm in pseudo-code.

3 HTTP/1.1 Accept Header Fields

When a client tries to retrieve a resource from a web server using HTTP, it sends a request to the server. This request contains some information known as an Accept header. Currently HTTP/1.1 uses four Accept header fields to describe the capabilities and preferences of the client: `Accept`, `Accept-Charset`, `Accept-Encoding` and `Accept-Language`. The `Accept` field describes which MIME[14] types are accepted by a browser. MIME is an acronym for Multipurpose Internet Mail Extensions, a standard defined by the Internet Engineering Task Force (IETF)[15],[16] and controlled by the Internet Assigned Numbers Authority (IANA)[17]. The other Accept header fields describe preference for character set, encoding and language respectively. Here are two examples of the Accept header fields produced by different browsers:

Internet Explorer 5.0

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel,
application/msword, */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
```

Netscape Navigator 4.73

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

Although it does not directly affect the implementation presented here, it is important to note that neither of these browsers correctly obeys the HTTP/1.1 content negotiation standard. They both use wildcards to specify their `Accept` preferences and there are several MIME types missing, such as `text/html` and `text/plain`. Unfortunately this means that it is not possible for servers to perform server-based content negotiation. One possible explanation for this is that the companies producing the browsers are concerned about the size of the Accept headers required to fully implement server based content negotiation. If the browser listed all the MIME types it could interpret in the header, the Accept header would be very big. This could be an issue because an Accept header is sent as part of every request for a resource made by a client. Therefore browsers use wildcards in order to reduce the size of their Accept headers.

Under HTTP/1.1, browsers can use two additional parameters for content negotiation purposes: `q` and `mxb`. `q` represents the quality factor between 0 and 1. If omitted, 1 is assumed. This indicates the desirability of various possible alternative versions of an object. For example

```
Accept-Language: fr; q=1.0, en; q=0.5
```

indicates that French resources are preferred to English resources. `mxb` is the maximum size in bytes of a resource that is acceptable by the browser or the user. This can also be associated with different types in a similar way to the previous parameter. For example

```
Accept: image/jpeg; mxb=5000, text/html
```

indicates that browser only wants JPEG images smaller than 5000 bytes.

4 Representing Header Fields In CC/PP

It is beyond the scope of this report to provide an introduction to CC/PP so readers are referred to the CC/PP specification[6] and the RDF specification[18]. Instead this section will describe the design decisions involved in creating a CC/PP vocabulary that can express HTTP/1.1 Accept headers. In essence CC/PP profiles consist of several sections known as components describing client attributes. The names and meaning of these components and attributes are dependent on the particular CC/PP vocabulary in use. For example the vocabulary for PCs currently described in the CC/PP specification is different to the vocabulary for WAP devices described in the UAProf specification. On a PC, typical components could be `HardwareProfile`, `SoftwareProfile` and `TerminalBrowser` whereas on a WAP phone they will be `HardwarePlatform`, `SoftwarePlatform`, `BrowserUA`, `WAPCharacteristics`, `NetworkCharacteristics` and `PushCharacteristics`. UAProf includes the attributes `CcppAccept`, `CcppAccept-Language`, `CcppAccept-Charset` and `CcppAccept-Encoding` that correspond to the fields in a HTTP request header. Attributes are located in specific components, so for example `CcppAccept` is located in `BrowserUA`. The implementation described here provides support for different vocabularies via an external XML configuration file. This will be discussed in more depth in a later section.

Secondly it was necessary to come up with a more complex way of grouping attribute data than described in the examples in the CC/PP specification. CC/PP attribute data comes in two types: simple literal text values such as URI's, text, integer numbers and rational numbers or complex sets of values expressed using the RDF bag construct. As Accept header attributes consist of a set of values they are naturally described using RDF bags. However in order to accurately mimic HTTP/1.1 content negotiation it is necessary to express preferences for different file types (`q` values) and maximum resource sizes (`mxb` values). In CC/PP this can be done by making the Accept type, `q` value and `mxb` value all attributes of a single anonymous node as shown in Figure 1.

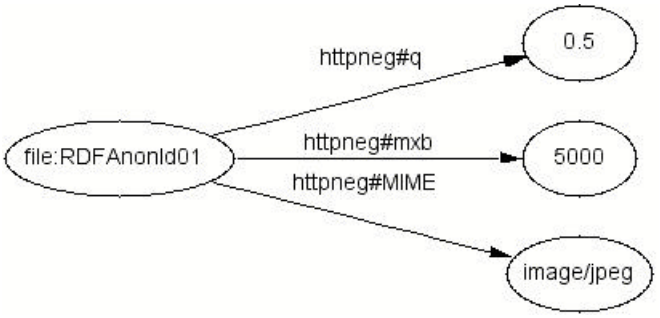

Figure 1 - Using Anonymous nodes in CC/PP

The UAProf specification does not allow anonymous nodes to be used in this way so it is not possible to express preferences for file types or maximum file sizes in the UAProf client profile. This is because although UAProf profiles are expressed in RDF they do not utilize its full descriptive power; rather they are best thought of as consisting of several tables, each table corresponding to a component, each containing a single tier of either attribute value pairs or attribute value sets. The implementation described here can detect whether anonymous nodes are being used to provide complex grouping of attribute data and hence determine whether it is dealing with an extended CC/PP or UAProf profile. In order to better understand this, consider the HTTP Accept header:

```
Accept: text/html; q=1.0, text/plain; q=0.8, image/jpeg; q=0.6
Accept-Language: fr; q=1.0, en; q=0.5
```

This Accept header can be represented by the RDF graph shown in Figure 2 and Figure 3. All the graphs generated in this report have been generated using the W3C's SiRPAC tool[19]. The profile shown in Figure 2 uses the UAProf component scheme. Each component is labelled using an `<rdf:type>` attribute as described in the UAProf and CC/PP specifications. The BrowserUA component is shown in more detail in Figure 3. It contains two bags, one for the `Accept` field and the other for the `Accept-Language` field. Specific object types are represented by anonymous nodes with attributes as described above.

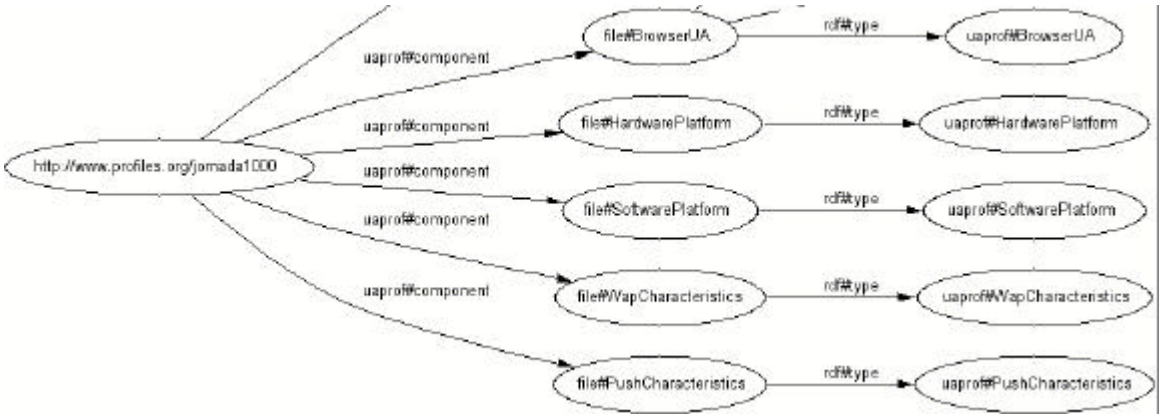

Figure 2 - CC/PP Profile : use of components

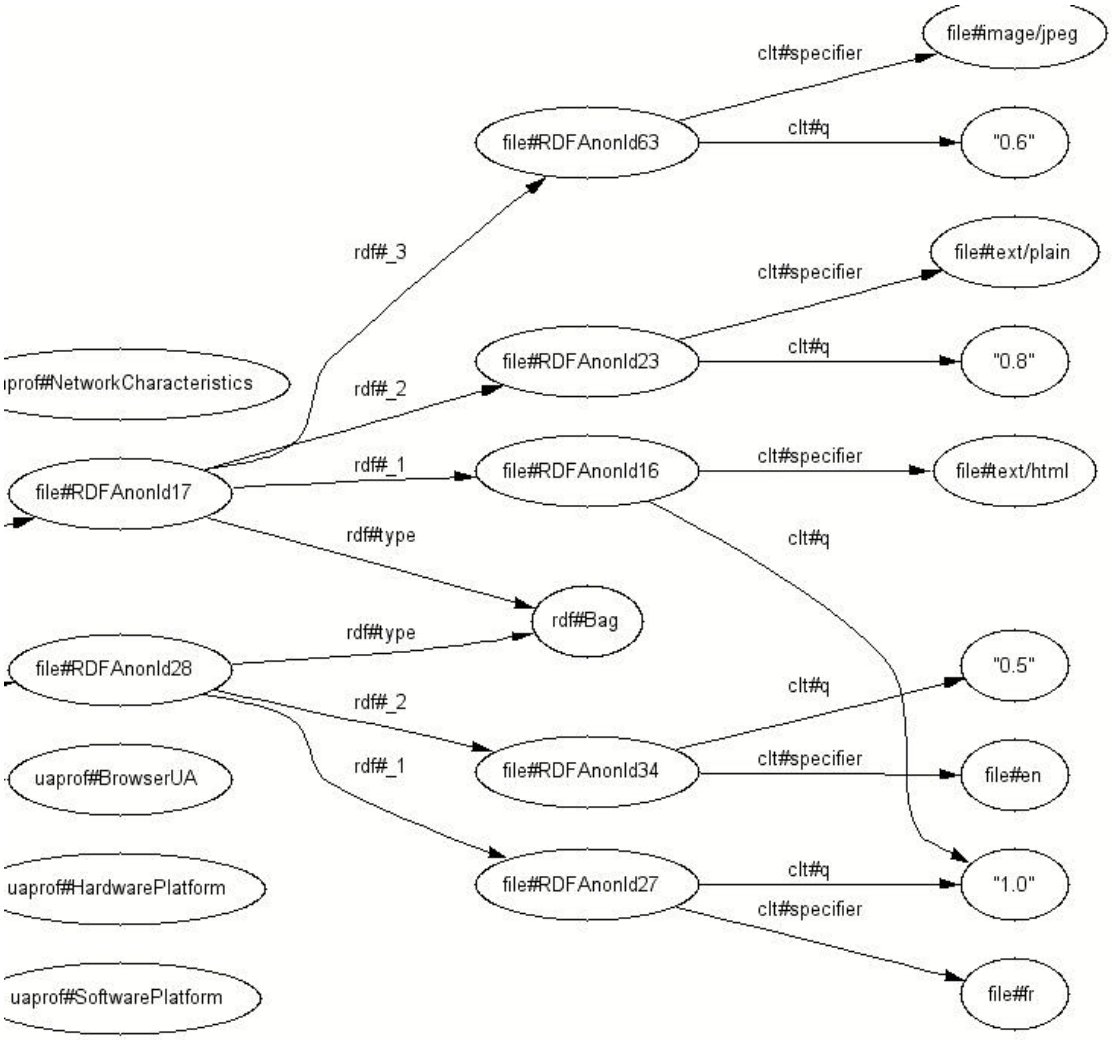

Figure 3 - CC/PP Profile : representation of Accept headers

The XML serialisation of the profile shown in Figure 2 and Figure 3 is as follows:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ccpp="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#"
    xmlns:clt="http://marks.profile.org/2001/05-clt#">
    <rdf:Description about="http://www.profiles.org/jornada1000">
        <ccpp:component>
            <rdf:Description ID="BrowserUA">
            <rdf:type  resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#BrowserUA"/>
                <ccpp:CcppAccept>
                    <rdf:Bag>
                        <rdf:li rdf:parseType="Resource">
                            <clt:specifier>text/html</clt:specifier>
                            <clt:q>1.0</clt:q>
                        </rdf:li>
                        <rdf:li rdf:parseType="Resource">
                            <clt:specifier>text/plain</clt:specifier>
                            <clt:q>0.8</clt:q>
                        </rdf:li>
                        <rdf:li rdf:parseType="Resource">
                            <clt:specifier>image/jpeg</clt:specifier>
                            <clt:q>0.6</clt:q>
                        </rdf:li>
                    </rdf:Bag>
                </ccpp:CcppAccept>
                <ccpp:CcppAccept-Language>
                    <rdf:Bag>
                        <rdf:li rdf:parseType="Resource">
                            <clt:specifier>fr</clt:specifier>
                            <clt:q>1.0</clt:q>
```

```
                        </rdf:li>
                        <rdf:li rdf:parseType="Resource">
                            <clt:specifier>en</clt:specifier>
                            <clt:q>0.5</clt:q>
                        </rdf:li>
                    </rdf:Bag>
                </ccpp:CcppAccept-Language>
            </rdf:Description>
        </ccpp:component>
        <ccpp:component>
            <rdf:Description ID="HardwarePlatform">
<rdf:type resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#HardwarePlatform" />
            </rdf:Description>
        </ccpp:component>
        <ccpp:component>
            <rdf:Description ID="SoftwarePlatform">
<rdf:type resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#SoftwarePlatform" />
            </rdf:Description>
        </ccpp:component>
        <ccpp:component>
            <rdf:Description ID="NetworkCharacteristics">
<rdf:type resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#NetworkCharacteristics" />
            </rdf:Description>
        </ccpp:component>
        <ccpp:component>
            <rdf:Description ID="WapCharacteristics">
<rdf:type resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#WapCharacteristics" />
            </rdf:Description>
        </ccpp:component>
        <ccpp:component>
            <rdf:Description ID="PushCharacteristics">
<rdf:type  resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#PushCharacteristics" />
            </rdf:Description>
        </ccpp:component>
    </rdf:Description>
</rdf:RDF>
```

5 Representing Variant Maps using RDF

The Apache Web Server currently uses files called *variant maps* to describe information about alternate variants. The variant map has an entry for each variant, describing the content type and optionally the content language, content encoding, content character set, the source quality and the file size. In addition variant preferences can be expressed using the source quality parameter `qs`. This could be used to identify a JPEG resource as being preferable to an ASCII art resource. For example the variant map

```
URI: foo

URI: foo.jpeg
Content-type: image/jpeg; qs=0.8

URI: foo.gif
Content-type: image/gif; qs=0.5

URI: foo.txt
Content-type: text/plain; qs=0.01
Content-Language: en
```

indicates that `foo.jpeg` is preferred to `foo.gif`. Hence source quality works in a similar way to the quality factor in the device profile.

For the purposes of the implementation it was decided to represent the variant map in RDF in order to simplify the negotiation process. The RDF graph for the variant map consists of a single bag that contains all the variants. Each variant is described by an

anonymous node with associated properties in a similar way to that used in the device profile. Part of the RDF graph is shown in Figure 4.

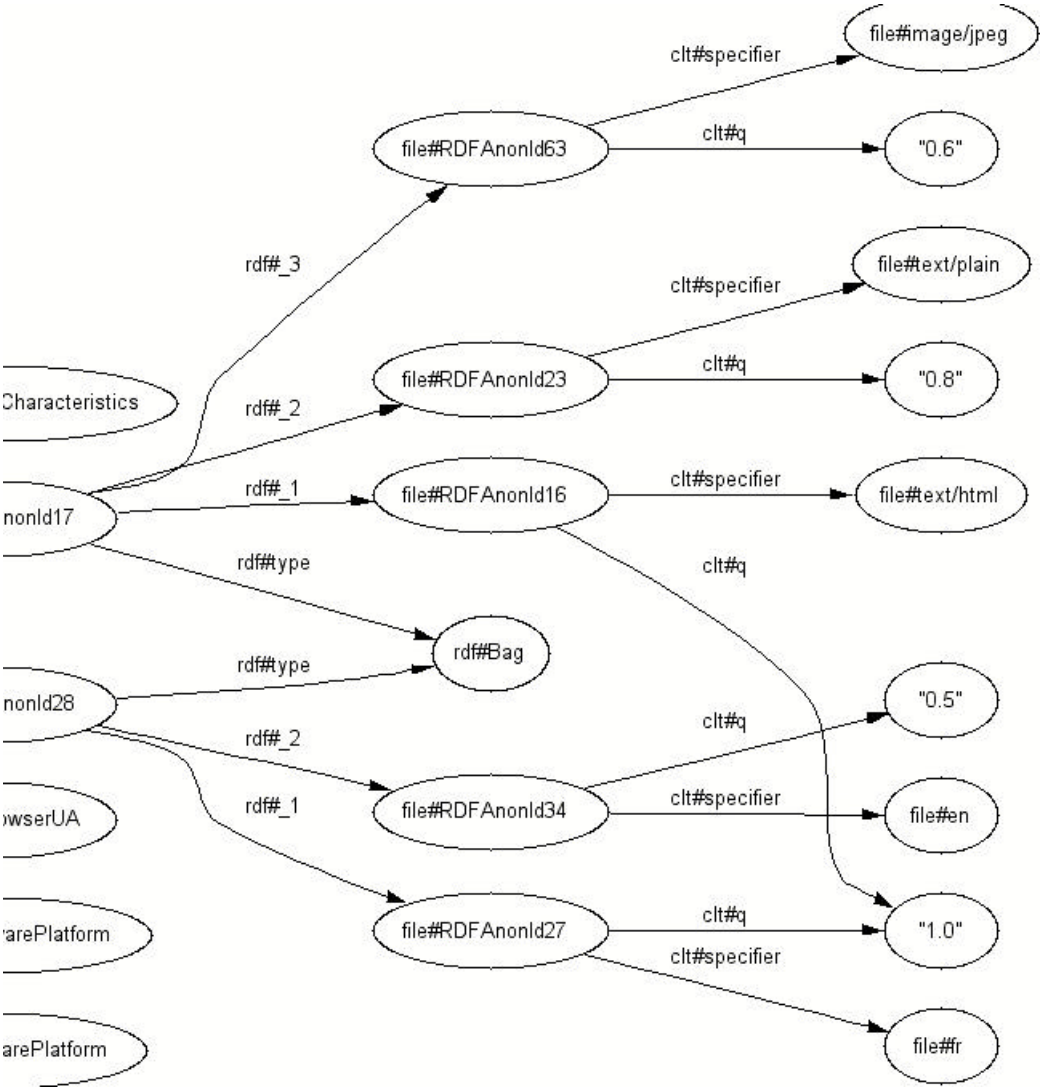

Figure 4 - Subgraph of RDF Representation of Variant Map

This is the XML serialisation of variant map:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:vrt="http://marks.profile.org/2001/05-vrt#"
xmlns:ccpp="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010426#">
    <rdf:Description about="http://www.mywebsite.com/foo">
        <vrt:variantList>
            <rdf:Bag>
                <rdf:li rdf:parseType="Resource">
                    <vrt:contentLoc>http://www.mywebsite.com/foo.jpg</vrt:contentLoc>
                    <ccpp:CcppAccept>image/jpeg</ccpp:CcppAccept>
                    <vrt:qs>0.8</vrt:qs>
                    <vrt:variantSize>3000</vrt:variantSize>
                </rdf:li>
                <rdf:li rdf:parseType="Resource">
                    <vrt:contentLoc>http://www.mywebsite.com/foo.gif</vrt:contentLoc>
                    <ccpp:CcppAccept>image/gif</ccpp:CcppAccept>
                    <vrt:qs>0.5</vrt:qs>
                    <vrt:variantSize>4000</vrt:variantSize>
                </rdf:li>
                <rdf:li rdf:parseType="Resource">
                    <vrt:contentLoc>http://www.mywebsite.com/foo.txt</vrt:contentLoc>
```

```
                    <ccpp:CcppAccept>text/plain</ccpp:CcppAccept>
                    <ccpp:CcppAccept-Language>en</ccpp:CcppAccept-Language>
                    <vrt:qs>0.1</vrt:qs>
                </rdf:li>
                <rdf:li rdf:parseType="Resource">
                    <vrt:contentLoc>http://www.mywebsite.com/foo.mpg</vrt:contentLoc>
                    <ccpp:CcppAccept>application/mpeg</ccpp:CcppAccept>
                    <ccpp:CcppAccept-Language>en</ccpp:CcppAccept-Language>
                    <vrt:qs>0.4</vrt:qs>
                    <vrt:variantSize>210000</vrt:variantSize>
                </rdf:li>
            </rdf:Bag>
        </vrt:variantList>
    </rdf:Description>
</rdf:RDF>
```

6 Negotiation

The negotiation algorithm consists of three phases: supplying default preference values to the capability profile where necessary, supplying default preferences values to the variant map where necessary and the actual negotiation proper. The negotiation is based directly on the algorithm used by the Apache Web Server with one refinement: it is possible to specify which Accept fields must be matched (hard constraints) and which will be matched if possible (soft constraints). For example if the Accept field is a hard constraint i.e. if a device only accepts WML files then if the server has no suitable alternate it will return nothing. The other fields are soft constraints i.e. if a device requests French alternate then a server will return a French alternate if one exists; otherwise it will return a resource in any language. Attribute fields can easily be configured as hard or soft constraints via the external XML vocabulary definition file. In order to better understand the negotiation algorithm, here is a pseudo-code description:

default client profile

```
foreach attribute type
       if the profile contains a bag for this attribute type
              foreach node in the bag
                     if node is a literal then
                            this is a UAProf profile, do nothing
                     else if node is a resource then
                            this is an extended CC/PP profile
                            if node does not have quality factor property
                               add quality factor property with default value
                            end if
                     end if
              next
       end if
next
```

default variant map

```
foreach variant
       if variant does not have source quality property
              add source quality property with default value
       end if
next
```

negotiate

```
add defaults to client profile
add defaults to variant map
for each variant
       variant size = 0
       get the source quality property of the variant
       if the variant has a size property
              get the size property of the variant
```

```
        end if
        preference = matchAxesOfNegotiation()
        if preference is greater than highest preference found so far
                this variant becomes preferred variant
        end if
next
```

match axes of negotiation

```
clear collection of maxsize constraints
for each variant property
        if the property is an attribute
                if attribute is a hard constraint then
                        clientpref = 0 as must match this in profile
                else
                        clientpref = 1 as match not essential
                end if
                if the profile contains a bag for this attribute
                        for each node in bag
                                clientpref = matchnode(node)
                else if the profile contains a single value for this attribute
                        clientpref = matchnode(attributenode)
                end if
                pref = pref * clientpref
        end if
next
if variantSize > any maxsize constraint
        pref = 0
end if
```

matchnode

```
if the node is a resource
        this is an extended CC/PP profile
        if the node matches the variant attribute
                client pref = quality property of node
                if the node has an mxb property
                        add mxb to the maxsize collection
                end if
        end if
else
        this is a uaprof profile
        if the node matches the variant attribute
                clientpref = 1
        end if
end if
```

7 Configuring CC/PP Vocabularies

As noted previously, different CC/PP vocabularies use different component names and different client attributes. In order to simplify the process of adapting the negotiation algorithm to different vocabularies, it uses an XML configuration file called `vocab.xml` shown below. The file contains component elements corresponding to components and negaxis elements corresponding to attributes. The name negaxis was chosen to avoid confusion with XML attributes. In the file both the component name and attribute name are configured using the name attribute of the appropriate element. The file also indicates if client attributes are sets (like `CcppAccept`) or single values (like `ColorCapable`) using the `acceptbag` attribute. Finally it is possible to configure if the attribute is a hard constraint or a soft constraint using the `mustmatch` attribute. The `acceptbag` and `mustmatch` attributes both accept Boolean values. For example here is a configuration file for basic HTTP/1.1 content negotiation:

```
<?xml version="1.0" encoding="UTF-8"?>
<ccppVocab>
    <component name="BrowserUA">
        <negaxis name="CcppAccept" mustmatch="true" acceptbag="true"/>
        <negaxis name="CcppAccept-Charset" mustmatch="false" acceptbag="true"/>
        <negaxis name="CcppAccept-Language" mustmatch="false" acceptbag="true"/>
        <negaxis name="CcppAccept-Encoding" mustmatch="false" acceptbag="true"/>
    </component>
    <component name="HardwarePlatform"/>
    <component name="SoftwarePlatform"/>
</ccppVocab>
```

and this is a longer configuration file suitable for a UAProf device:

```
<?xml version="1.0" encoding="UTF-8"?>
<ccppVocab>
    <component name="BrowserUA">
        <negaxis name="CcppAccept" mustmatch="true" acceptbag="true"/>
        <negaxis name="CcppAccept-Charset" mustmatch="false" acceptbag="true"/>
        <negaxis name="CcppAccept-Language" mustmatch="false" acceptbag="true"/>
        <negaxis name="CcppAccept-Encoding" mustmatch="false" acceptbag="true"/>
        <negaxis name="FramesCapable" mustmatch="true" acceptbag="false"/>
        <negaxis name="TablesCapable" mustmatch="true" acceptbag="false"/>
    </component>
    <component name="HardwarePlatform">
        <negaxis name="ScreenSize" mustmatch="false" acceptbag="false"/>
        <negaxis name="ImageCapable" mustmatch="true" acceptbag="false"/>
        <negaxis name="BitsPerPixel" mustmatch="false" acceptbag="false"/>
        <negaxis name="ColorCapable" mustmatch="true" acceptbag="false"/>
    </component>
    <component name="SoftwarePlatform"/>
    <component name="NetworkCharacteristics"/>
    <component name="WapCharacteristics"/>
    <component name="PushCharacteristics"/>
</ccppVocab>
```

8 Conclusions

This technical report has described an implementation of a content negotiation algorithm based on HTTP/1.1 that works with CC/PP and UAProf. Creating this implementation has been very instructive as there are currently no freely available examples of algorithms that process CC/PP profiles. This investigation has highlighted a number of issues that will be discussed in this section.

8.1 Negotiation algorithms must be able to deal with extensible vocabularies

The implementation described here has one big advantage over HTTP/1.1 content negotiation: it is extensible. For example HTTP/1.1 content negotiation is insufficient for device independence as devices of different types (e.g. PDAs and PCs) might accept the same MIME type but require different resources (e.g. a PDA requires a smaller image). In the implementation described here it is easy to add a new attribute using the `vocab.xml` file (for example `DeviceClass`) with several values (for example `HandHeld` and `PC`) that can be used to select the appropriate variant of a resource. Extensibility is a necessity for any negotiation algorithm as it is likely that there will never be a single CC/PP vocabulary for device independence particularly as we already have a legacy profile in the form of UAProf. Therefore negotiation algorithms must be able to cope with multiple vocabularies. One way to achieve this is to use external configuration files.

8.2 Vocabularies must be well designed

Despite the likelihood of a proliferation of profile vocabularies, it is desirable that device manufacturers use a small number of carefully designed and standardized vocabularies. Furthermore these vocabularies need to be sufficiently flexible to represent not just the device capabilities but also user preferences. Therefore it is proposed that there is a need for more work both on vocabularies and on profile processing in order to come up with some guidelines for vocabulary creators. For example the negotiation algorithm implemented here is fairly simple but it was not possible to implement the full negotiation algorithm using the UAProf vocabulary. This was because all attributes in a UAProf profile, whether simple or complex, can only have a single item of associated data. This makes it difficult to specify preferences as this requires the attribute value and its associated preference value. There are ways in UAProf of associating more than a single value with an attribute: for example in UAProf the attribute `ScreenSizeChar` uses `x` as a separator between two parameters e.g.

```
<prf:ScreenSizeChar>15x6</prf:ScreenSizeChar>
```

however such approaches mean that the profile is no longer truly XML readable as non-XML separators are used.

8.3 More complex matching algorithms may be necessary

One problem with negotiation algorithms based on HTTP/1.1 is that the matching process is too simplistic. Firstly in the current framework although devices can have multiple values for attributes in profiles, resources cannot have multiple values for attributes in variant maps. This could cause problems: for example a resource might be viewable by devices where the `AcceptLanguage` attribute is either `EN-US` or `EN-GB` as both devices are specifying a preference for English documents. Secondly more flexible methods of matching then simple equality is needed. The IETF media feature set method of content negotiation used relational operators such as less than, greater than, less than or equals etc. so that resources can specify the constraints that are necessary for a resource to be displayed. For example a resource might require a device with a screen bigger than 640 pixels in width in order to display a particular image.

8.4 More complex ways of grouping attributes may be necessary

The IETF proposal also allows client attributes to be described in a more complex way than the algorithm described here. For example it is possible to express that certain attributes of a device may be associated with certain media types or certain modes e.g. a device might be able view streamed video resources up to 640 x 480 pixels in size and JPEG images up to 1024 x 768 pixels in size. In media feature sets, capabilities like this are expressed by linking attributes using ANDs and ORs. Early working drafts of the CC/PP specification described how this might be done in CC/PP although recent versions of the specification do not contain such examples. This additional expressive power comes at a price though: client and resource profiles will be longer and more complicated and the negotiation algorithm may need to manipulate the structure of the profiles using rules for simplifying Boolean logic in order to perform matching. Further investigation of processing profiles is necessary to

understand how complex profiles need to be in order to support adaptation of content to multiple devices.

8.5 Content authors will not write XML serialised RDF by hand

Although using RDF to represent the variant maps makes sense for the negotiation algorithm, RDF serialisations are much more verbose and cumbersome for the content author to edit than the text variant maps used in Apache. This problem could be resolved by creating a tool that assists the user in the creation of these maps. Another approach could be to use a different XML serialisation which is less verbose.

8.6 Support for legacy devices and software is needed to speed up the adoption of CC/PP

Currently there are no devices or browsers available that support CC/PP, although the next generation of WAP devices will support UAProf. This is a barrier to the uptake of CC/PP as there is no point in adding support for it to servers until a sufficient number of devices support it. One way round this is to develop a CC/PP repository that contains profiles for legacy devices and browsers, so that when a device does not support CC/PP, the user agent string is used to retrieve the corresponding profile from the repository. Such a scheme has limitations as it does not support user personalisation as all devices of the same type have the same profile. However such a repository may be essential in order to demonstrate the utility of CC/PP.

8.7 The negotiation algorithm does not implement the entire CC/PP specification

The implementation presented here is incomplete in several ways: it does not support the use of defaults in CC/PP profiles or the use of the CC/PP proxy vocabulary. Only the negotiation algorithm is implemented, not the CC/PP exchange protocol. Finally the implementation only performs the negotiation, it does not actually retrieve any content. These issues have been ignored because the focus of the implementation is primarily to inform on the design of negotiation algorithms based on CC/PP rather than provide a full working prototype. Once negotiation is well understood, the intention is to rectify these omissions.

8.8 Content negotiation is not the only way to process device profiles

Finally it is important to note content negotiation is not the only way that CC/PP profiles may be used to support device independence. Alternative approaches to processing include making profile attributes available to XSLT stylesheets so the stylesheets can adjust the transform based on information in the profile. A different approach would use profile attributes for media transcoding for example converting images to a specific size and format on the fly for a device.

9 Appendix A: Content Negotiation API

The implementation uses two public classes, CcppProfile and VariantMap, that are both subclasses of ModelMem in Jena. CcppProfile is an API for creating CC/PP profiles whereas as VariantMap is an API for creating variant maps and performing

negotations using a variant map. The public methods available in the classes are shown in the tables below. For more information, see the JavaDoc files.

CcppProfile
+ Public CcppProfile(String namespace) + Public CcppProfile(FileReader theFile) + Public boolean add(String attribute, String value, double q, int mxb) + Public boolean add(String attribute, String value, double q) + Public boolean add(String attribute, String value, int mxb) + Public boolean add(String attribute, String value) + Public void defaultPreference()

VariantMap
+ Public VariantMap(String mapName) + Public void add(String resourceName, String[] neglist, String[] type, double qs, int variantsize) + Public void add(String resourceName, String[] neglist, String[] type, double qs) + Public void add(String resourceName, String[] neglist, String[] type, int variantsize) + Public void add(String resourceName, String[] neglist, String[] type) + Public void defaultPreference() + Public void negotiate(CcppProfile theProfile)

For example the following Java code will create a CC/PP profile:

```
CcppProfile myprof = new CcppProfile("http://www.profiles.org/jornada1000");
myprof.add("CcppAccept", "text/html", 1.0);
myprof.add("CcppAccept", "text/plain", 0.8);
myprof.add("CcppAccept-Language", "fr", 1.0);
myprof.add("CcppAccept-Language", "en", 0.5);
```

Whereas the following Java code will create a variant map:

```
VariantMap variants = new VariantMap("http://www.mywebsite.com/foo");
String[] firstType = {"image/jpeg"};
String[] secondType = {"image/gif"};
String[] thirdType = {"text/plain", "en"};
String[] fourthType = {"application/mpeg", "en"};
String[] firstNeg = {"CcppAccept"};
String[] thirdNeg = {"CcppAccept", "CcppAccept-Language"};
variants.add("http://www.mywebsite.com/foo.jpeg", firstNeg,
              firstType, 0.8, 3000);
variants.add("http://www.mywebsite.com/foo.gif", firstNeg,
              secondType, 0.5, 4000);
variants.add("http://www.mywebsite.com/foo.txt", thirdNeg,
              thirdType, 0.1);
variants.add("http://www.mywebsite.com/foo.mpg", thirdNeg,
              fourthType, 0.4, 210000);
```

In addition to the two classes `CcppProfile` and `VariantMap`, there is an additional helper class `VocParse` that provides vocabulary information. It supplies a number of constants that define some relevant namespaces and property names. It also supplies a number of methods for accessing the data structure that is created when the XML vocabulary file is parsed when the object is created. These methods can return the number of components and attributes, accessor functions to get the names of the components and attributes, as well as accessor functions to determine whether an

attribute is a hard constraint, a bag as well as which component an attribute belongs to.

10 Appendix B: Test Plan

A test harness was created in order to perform simple tests on the API. The test plan is as follows:

1. Create a CC/PP profile equivalent to the following Accept header:

   ```
   Accept: text/html; q=1.0, text/plain; q=0.8, image/gif; q=0.6
   Accept-Language: fr; q=1.0, en; q=0.5
   ```

   Then produce an XML serialisation of this profile, export it to SiRPAC and validate the resulting RDF graph.
2. Create a CC/PP profile equivalent to the following Accept header:

   ```
   Accept: text/html; q=1.0, text/plain; q=0.8; text/rtf; image/jpeg;
   Accept-Language: fr; q=1.0, en; q=0.5, jp; kr;
   ```

   Then apply the default preferences method and produce an XML serialisation of this profile, export it to SiRPAC and validate the resulting RDF graph.
3. Create an RDF variant map equivalent to the following text file:

   ```
   http://www.mywebsite.com/foo.jpeg
   Content-type: image/jpeg; qs=0.8
   Content-size: 3000

   http://www.mywebsite.com/foo.gif
   Content-type: image/gif; qs=0.5
   Content-size: 4000

   http://www.mywebsite.com/foo.txt
   Content-type: text/plain; qs=0.1
   Content-language: en

   http://www.mywebsite.com/foo.mpg
   Content-type: application/mpeg; qs=0.4
   Content-language: en
   Content-size: 210000
   ```

   Then produce an XML serialisation of this variant map, export it to SiRPAC and validate the resulting RDF graph.
4. Create a CC/PP profile equivalent to the following Accept header:

   ```
   Accept: text/html; q=1.0, text/plain; q=0.8, image/gif; q=0.6,
          mxb=6000
   Accept-Language: fr; q=1.0, en; q=0.5
   ```

   and a variant map as described in test 3. Perform a negotiation and verify that the `foo.gif` variant is selected.
5. Create a CC/PP profile equivalent to the following Accept header:

   ```
   Accept: text/html; q=1.0, text/plain; q=0.8, image/*; q=0.6,
          mxb=6000
   Accept-Language: fr; q=1.0, en; q=0.5
   ```

and a variant map as described in test 3. Perform a negotiation and verify that the `foo.jpeg` variant is selected.

6. Load the example CC/PP profile described in[11] and create the variant map described in test 3. Perform a negotiation and verify that the `foo.gif` variant is selected.
7. Create a CC/PP profile as described in test 1, but add an additional single value UAProf attribute called BitsPerPixel with a value of 2. Then produce an XML serialisation of this profile, export it to SiRPAC and validate the resulting RDF graph.
8. Create an RDF variant map equivalent to the following text file:

```
http://www.mywebsite.com/foo.jpeg
Content-type: image/jpeg; qs=0.8
Content-size: 3000
ColorCapable: No

http://www.mywebsite.com/foo.gif
Content-type: image/gif; qs=0.5
Content-size: 4000
ColorCapable: No

http://www.mywebsite.com/foo.wbmp
Content-type: image/vnd.wap.wbmp; qs=0.4
Content-size: 2100
ColorCapable: No
```

Here `ColorCapable` means that the resource requires a device that matched that `ColorCapable` attribute i.e. the GIF resource can only be displayed on a device for which `ColorCapable` is yes. Then perform a negotiation using the CC/PP profile described in[11] and verify that the `foo.wbmp` variant is selected.

---

[1] Mark Butler, Current technologies for device independence, HPL-2001-83, http://www.hpl.hp.com/techreports/2001/HPL-2001-83.html
[2] W3C Web Accessibility Guidelines, http://www.w3c.org/WAI/
[3] RFC 2616: HTTP 1.1 Content Negotiation, page 70-73, ftp://ftp.isi.edu/in-notes/rfc2616.txt
[4] RFC 2533: A Syntax for Describing Media Feature Sets, ftp://ftp.isi.edu/in-notes/rfc2533.txt
[5] Richard Blaylock, Browser Detection, http://hotwired.lycos.com/webmonkey/99/02/index2a.html
[6] Composite Capabilities / Preferences Profile Structure and Vocabularies, W3C, http://www.w3.org/TR/CCPP-struct-vocab/
[7] Wireless Application Forum, http://www.wapforum.org/
[8] Resource Description Framework, http://www.w3.org/RDF/
[9] Jena RDF Framework, http://www-uk.hpl.hp.com/people/bwm/rdf/jena/index.htm
[10] Apache HTTP Server Content Negotiation, http://httpd.apache.org/docs/content-negotiation.html
[11] WAP Forum WAG UAProf Draft Version 02-May-2001, WAP-248-UAPROF-20010502, http://www.wapforum.org/
[12] HTTPCCPP source code, http://www-uk.hpl.hp.com/people/marbut/httpccpp.zip
[13] HTTPCCPP licence, http://www-uk.hpl.hp.com/people/marbut/httpccpplicense.txt
[14] Jeffery Dwight and Michael Erwin, Using MIME with CGI, http://sunsite.net.edu.cn/tutorials/se_cgi/Cgi10fi.htm
[15] IETF RFC 1341, ftp://ftp.isi.edu/in-notes/rfc1341.txt
[16] IETF RFC 1521, ftp://ftp.isi.edu/in-notes/rfc1521.txt
[17] IANA Mime Type Registry, ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/media-types
[18] Resource Description Framework Model and Syntax Specification, http://www.w3.org/TR/REC-rdf-syntax/
[19] SiRPAC, http://www.w3.org/RDF/Implementations/SiRPAC/