Adapted Content Delivery for Different Contexts

Tayeb Lemlouma, Nabil Layaïda

To cite this version:

Tayeb Lemlouma, Nabil Layaïda. Adapted Content Delivery for Different Contexts. Proc. International Symposium on Applications and the Internet (SAINT 2003), Jan 2003, Orlando, FL, United States. pp.190-197. inria-00423426

HAL Id: inria-00423426

https://hal.inria.fr/inria-00423426

Submitted on 9 Oct 2009

Adapted Content Delivery for Different Contexts

Tayeb Lemlouma[1] and Nabil Layaïda[2]
OPERA Project, INRIA Rhône Alpes
[1]*Tayeb.Lemlouma@inrialpes.fr* [2]*Nabil.Layaida@inrialpes.fr*

Abstract

In this paper, we present a framework which allows adapted content delivery for different target contexts. This framework is based on a Universal Profiling Schema UPS for describing the environment characteristics and on an profile exchange protocol. In the server and the proxy side, we give a strategy for matching the different constraints (clients, servers, content, etc.) in order to find an agreement between the server adaptation capabilities and the client preferences and constraints. Usually such environments are subject to frequent changes. To tackle this difficulty, we propose a dynamic adaptation approach based on XSLT for structural transformation and resource aware transcoders for the media adaptation.

1. Introduction

In the last few years, new devices such as small palm computers, smart phones, pocket PCs became common components of the computing infrastructure. According to some estimates cited by the W3C, by the near future, 75% of the web content access will be soon generated by these devices rather than by desktop PCs.

At the same time, content delivery practices faces new challenges regarding exchange protocols and the accompanying strategies used to meet client, server and network constraints [10]. In particular, the diversity in the current infrastructure is still increasing at every level: users, network, access methods, protocols, etc.

In such heterogeneous environments, it's clear that transformation have a particularly important role to play in the content delivery. In addition, users needs vary with respect to their capacities and preferences, therefore, content servers can not perform adaptations, in advance, for every kind of client. This underlines the need of adapting the content dynamically.

Many solutions and architectures have been proposed to help in designing adaptive multimedia systems for heterogeneous devices. In general, the focus is given on taking into account some predefined constraints and working to optimize and deliver the content. For example, by transforming HTML content to some markup languages [15]. Some studies aim to handle network limitations by reducing the bandwidth consumption [1], or the device constraints by handling videos streams in terms of spatial and temporal transcoding [9] and adapting the audio quality to the network characteristics [2], etc.

Unfortunately, these systems do not address neither client and environment modeling and description nor exchange protocols which allow the delivery of an adapted content. Furthermore, the proposed content adaptation strategies are limited to specific applications and can not be applied to larger scales such as the web.

In this paper, we present a strategy for adapted content delivery in heterogeneous environments. This strategy is described in terms of description model, communication protocol and negotiation and adaptation methods. It is applied in the context of multimedia content delivery for resource limited devices in an environment subject to variable constraints. We define adaptation methods which allow take into account the environment constraints and end with a service agreement between the server, the network capabilities and the client requirements.

2. The client context and the UPS schema

Before discussing the UPS description approach, let's have first a look on the user context description using the HTTP protocol and the servers behavior to these kink of requests. The following figure shows the request of a PDA device (iPAQ 3600) using Pocket Internet Explorer under Windows CE. The requested content is a GIF image.

```
GET http://www.inrialpes.fr/opera/people/Tayeb.Lemlouma/E-Learning-Arch.gif HTTP/1.1
Accept: */*
UA-OS: Windows CE (POCKET PC) - Version 3.0
UA-color: color16
```

```
UA-pixels: 240x320
UA-CPU: ARM SA1110
UA-Language: JavaScript
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/2.0 (compatible; MSIE 3.02; Windows CE;
240x320)
Host: www.inrialpes.fr
```

Figure 1. The HTTP request of a PDA (iPAQ 3600) using Pocket IE under Windows CE

As we can see through the previous examples, the player includes the client context description as header fields inserted into the request. This description depends on the used device and varies from a player to another. Unfortunately, most of the existing servers don't take into account this kind of information. The following figure shows the requested image (Figure 1) as displayed by the PDA.

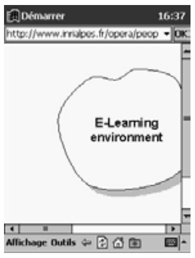

Figure 2. The received reply on a PDA device

Here, the client context wasn't usefully considered by the server since the display capabilities were not taken into account. As shown earlier, context information were conveyed inside the request using the two HTTP header fields: "UA-color: color16" and "UA-pixels: 240x320". As we can see, the received image is not compatible with it display limitations. The image can't be displayed entirely on the screen. This kind of situations becomes more complicated when the image is included in complex documents structures such as in a HTML page.

A context is defined generally as: 'Any information than can be used to characterize the situation of any entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves' [3]. Other definitions exist such as defining the context as: 'the application's settings' [14].

We define the context from the content adaptation point of view as the set of information that has a direct relationship to the requested document. As context description is also a matter of details level, determining the information to provide is the responsibility of the exchange protocol used in contexts transmissions. In order to achieve the content adaptation and negotiation task, the content provider should request a sub set of the user context which is related to the functionalities of the content to be delivered. For example, the server can request the client context capabilities related to displaying videos if the original content contains videos. However, for performance reasons, it should avoid such description requests if the original content contains only text data.

The universal profiling schema (UPS) [11] was defined to serve as a universal model that provides a detailed description framework for different contexts. UPS is built on top of CC/PP [6] and RDF [13].

The description of a context in UPS is profiles stored as XML markup. UPS identifies three main categories of contexts: the client, the server and the network. The client category includes the *client profile* that describes the client capabilities and preferences in general and the *client resource profile* that describes the client capabilities for a particular resource. The server category includes the *document instance profile* that describes the document characteristics, the *resource profile* that describes a media resource and the adaptation method profile that describes an available adaptation method available in the server or a proxy side. The network category is represented by the *network profile* that describes the network characteristics.

The UPS approach of describing a context category using different profile types has many advantages. The approach allows minimizing the profiles size by separating the information according to its type. This favors re-use for example if the profile describes an image used in many documents. Furthermore, it allows optimizing the exchanged information between the server and the client: the server can receive a profile and request only the related profiles that are involved in the content delivery. Profiles are requested on demand using the profile links included in received profile. For example, the client profile includes usually profile links to resource client profiles. A package that ensures the creation of valid UPS profiles is given in [16].

3. Negotiation exchange protocol

The negotiation protocol is a set of adaptation and negotiation messages exchanged between the client and the server. These messages allow the transmission of the client description (client profile), context change, and profile requests, etc. The goal of such a protocol is to define a way of conveying clients' characteristics and transporting only the required information when necessary. Client characteristics will be used during the content adaptation and negotiation matching applied at the server side. The protocol allows to find an agreement between the capability of the server (and/or the proxies) and the client needs.

In our approach, we distinguish communication and negotiation aspects in the protocols. The HTTP [12] protocol is an example of a communication protocol. The negotiation protocol carry information related to the content negotiation and adaptation.

One possible approach is to integrate the two in a single protocol. The HTTP/1.0 and HTTP/1.1 [8] represent an example of such protocols. Using HTTP/1.0, the client context is sent inside the user agent request through a set of accept headers. The content negotiation is applied at the server side and consists in applying content variant selection. The client description in HTTP/1.1 is achieved in a similar way. However, the server applies transparent content negotiation strategy [8] by sending the list of available variants and their properties to the client. Here, the responsibility of selecting the best variant is left to the user agent.

Another exchange protocol is presented in [7]. The protocol, called CC/PP exchange protocol, defines a way for exchanging clients' description based on the HTTP extension framework [5] and complies with HTTP/1.1 [4]. The protocol uses mainly two concepts: the extension declaration and a set of different header fields. The strategy consists to send the client request with the profile information using URIs and profiles differences principle [7]. Unfortunately, the defined protocol depends widely to HTTP which limits the exchange of adaptation and negotiation messages to clients that use only the HTTP protocol.

From the adaptation point of view, the HTTP protocol presents two main limitations:

1) *A poor context description*: sending the context as several accept headers in every request is highly inefficient. Furthermore, the syntax of the user agent capabilities and preferences is not extensible and expressive to encompass cover all the diversity of devices and media resources.

2) *A limited adapted content delivery*: at the server side, the protocol uses a content negotiation strategy to provide an adapted content to the client. The applied strategy is based on the provision of several versions of the same content identified using a single URI. The process consists in matching the available versions properties with the client capabilities. Following such an approach requires providing the variants for each target context which puts the burden on content providers.

The exchange protocol use in our case is not dependent on the communication protocol. The protocol is optimized and only useful information is transmitted. The protocol defines the following minimal set of message types:

GET_GLOBAL_PROFILE
OK_SENDING_PROFILE
OK_SENDING_CHANGE
NO_PROFILES_CHANGE
NO_PROFILE_ACQUISITION

These messages are exchanged between a module located on the client and another on the server. The first message type is used by the server to get the current user agent context. The context is represented by a set of client properties written in UPS [11]. The server sends this request before the delivery to take into account the context in which the content will be used by the client. The second message type is used by the client or the intermediate proxy in order to send the profile requested by the server. During content access, client description (especially the preferences) can have some changes, the third message type is used to send theses changes. The fourth message type is used when there is no difference between the current client context and the last context sent to the server. Finally, the last message type is used when the user module is not able to send a description of the client.

The server handling of the possible changes that may occur into the context represents an important aspect. Here, we have two solutions: sending the complete context description with the context change or sending only the change (the profile difference). The first solution is not efficient. The second solution requires server profile caching and assumes that at least the initial context is already sent.

The protocol that we have defined uses a simple caching strategy based on the clients IP addresses. The user context is stored temporarily in the server (or the proxy) side and is associated to a unique name using the client IP address. Any further changes will be referred to the same user context name.

The defined protocol has been implemented in a proxy-based and server-based architecture. In the first situation the negotiation module is integrated to an intermediate proxy that handles the client requests and the server replies. A player listener module was developed to serve as communication proxy. The module receives players' requests and sends them to the original server. Client requests are sometimes modified according to the client context sent by the user context module (UCM).

The figure shows two sessions (between client UCM modules and the UCM listener) in which only negotiation-oriented messages are exchanged. The other sessions between players and the player listener (the proxy) are achieved using the communication protocol. The proxy-based architecture allows testing the context exchange and the content adaptation but doesn't allow a full control of the original server content.

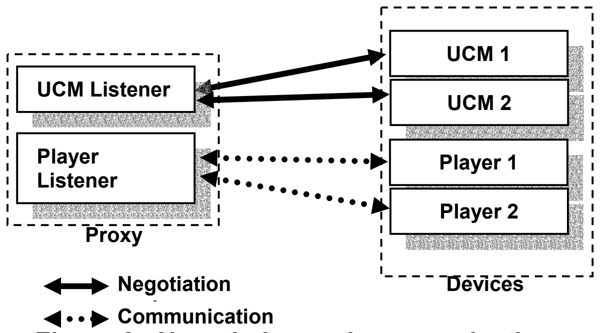

Figure 3. Negotiation and communication sessions

For example, in such architectures, the content selection and substitution can't be applied by the protocol because the proxy hasn't access to the entire server content. This module has also been integrated a server-based architecture. The player listener is located on the content server.

The global scenario can be summarized as follows:

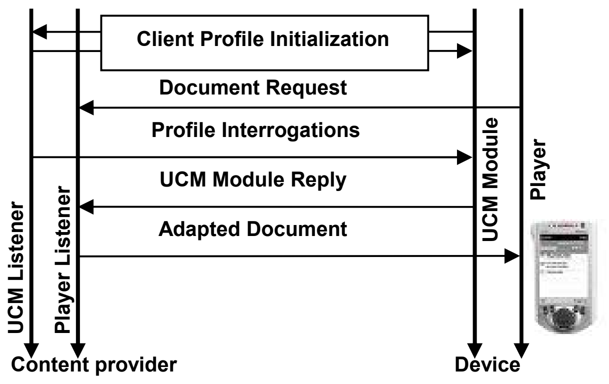

Figure 4. The general scenario between the device and the content provider

First the user context (or the profile) is initialized, and when the client requests a content, the UCM listener tries to exchange negotiation messages in order to retrieve the required information. Once, the content provider has all the required information, the content negotiation and adaptation task is executed and the results are sent to the client using the communication protocol.

4. The content negotiation strategy

Interactions with the content server can vary from a client to another. A client requests the content using its device (PDA, laptop, phone, etc.) with specific characteristics and capabilities. Consequently, the retrieved content will differ consequently. Content negotiation aims to guide content servers to deliver the appropriate content according to the user context, i.e. the client capabilities and the user preferences.

Generally, a content negotiation solution requires the following basic components (Figure 5):

a) A description tool of the context in which the content is used: such as the description of the client context, the server capabilities, the document profile, etc.

b) An exchange protocol: a well determined format for the exchange of control messages and the communication of the user context to the server or other entities.

c) Adaptation methods and content versions: used to adapt the content or to deliver a variant.

d) A matching strategy: an algorithm which is applied at the server side and which aims to match the different profiles (clients, document, server, etc.) in order to determine the best common context and methods to apply for the content delivery.

Content negotiation techniques are applied mainly following two ways:

1- **Variant selection**: consists to choose the best variant of the server content on behalf the user agent. The selection is applied on the available variant list and based on variants description and the user requirements. Selection parameters include the language, the media type, the char-set, etc. The decision of the selection can be determined using an algorithm that handles the different situations [12][8]. Unfortunately, variant selection is not enough flexible and dependents on the variants availability.

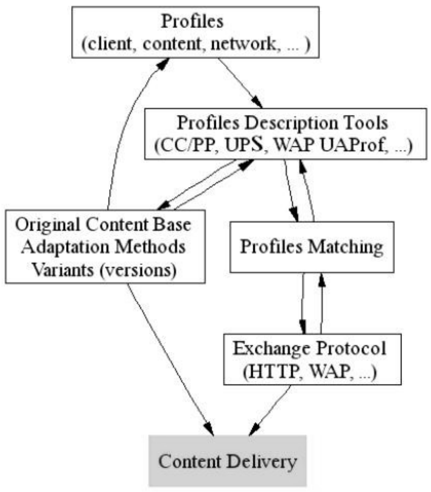
Profiles (client, content, network, ...)
Profiles Description Tools (CC/PP, UPS, WAP UAProf, ...)
Original Content Base Adaptation Methods Variants (versions)
Profiles Matching
Exchange Protocol (HTTP, WAP, ...)
Content Delivery

Figure 5. The content negotiation framework

2- **Content adaptation**: In several situations, the available content can't satisfy the client needs. In such cases, the content can be made available after applying some adaptations. The adaptation process can be achieved with a program, a script, a XSLT style sheet, etc. Adaptation techniques can be separated into two categories:

a) ***Media resources transformation category***: in this category, we group the media adaptation techniques like image and video transcoding (color reduction, resizing, etc.).

b) ***Structural transformation category***: is related to transformations that are applied on the document structure. An example of such application: transforming HTML to WML, filtering HTML documents, transforming XML to SVG, etc. A structural transformation can either keep the same media resource used by the original document, filter it or require an external media resource transformation to adapt the media for the end user context.

The goal of the content negotiation and adaptation is to use server and proxies capabilities to respond, in the best way, to the global context requirements. These requirements can be described as a set of *constraints* that need to be solved.

The concept of constraints takes a high importance in the practical application of the content negotiation. In this context, we define a constraint as: "*an atomic requirement that belongs to the environment in which the final delivered content will be used*". In practice, the architecture can take into account the constraints collection of more than one element.

In our approach, the resolution strategy starts from the assumption that the original content is accepted and continues by adding the constraints progressively. Finally, it ends with a solution that determines the content to be delivered. Possible solutions include an empty content in the case where the constraints can not be solved. In this case, the server sends a negative reply to the client.

In order to achieve an efficient content negotiation solution, the environment constraints must be well expressed in order to reflect the picture of the delivery context. Constraints syntax must:

1- Provide enough expressiveness, in order to allow the description of what is required.
2- Offer an easy analysis for resolution strategies.
3- Avoid ambiguity: constraints must result in a unique solution.

Example: Let's assume that we have in the server side an original content in the form of XML document written in English. Let's have the two following constraints expressed in the natural language: C_1={the preferred language is French}, C_2={the only supported language is French}. In a situation where the global context is represented by the first constraint C_1, the content of the server can be delivered directly to the client. This case is equivalent to having no constraints at all. In a situation where we have C_2 as environment constraint, the original content can't be delivered directly to the client because the content doesn't respect the environment constraints.

In our approach, the client constraints that depend on capabilities are extracted directly from the HardwarePlatform, SoftwarePlatform and BrowserUA components from the UPS client profiles. In the RDF bags: OnlySupportedResources, PreferredSupportedResource and NonSupportedResources, we extract additional constraints related to the content in terms of capabilities and preferences. During the negotiation matching, the client profile and the document instance profile of the requested content are parsed and the set of included constraints are stored in memory according to their types. The server makes the reference to the document instance profile. According to its content, the server can retrieve - using the exchange protocol- the client resource profile [11] that corresponds to the resource used by the requested content. For example, the server retrieves the client resource profile of the WBMP images if the original requested document uses WBMP images.

The server checks then if the resource (media or document) is supported by the client or not. In the positive case the resource is sent directly to the client without any modifications. In the negative case the server checks if there is any other version that can respond to the client requirements. Links to the list of versions related to the resource are included in the document instance profile or the client resource profile [11]. If the server succeeds to find a variant that responds to the client requirements, the original resource is substituted by the variant resource; otherwise the server tries to adapt the original resource. To achieve this operation, the server compares the original resource description (using its profile) and the set of the input requirements of each available adaptation methods included in the RDF bag: InputRequirements of the adaptation method profile [11]. If the resource description matches the input requirements of an adaptation method, the server checks if the output description of this method (included in the RDF bag OutputDescription of the adaptation method profile) matches the client requirements. If yes, the server applies this adaptation method on the original resource and delivers the created resource to the client. In the negative case, i.e. no adaptation method can be applied, the server sends a negative reply concerning the requested resource in order to avoid the client blocking.

5. Architecture

The negotiation exchange protocol and the content negotiation and adaptation strategies have been implemented within a proxy-based and server-based architecture (Figure 6). In both architectures, the used network is based on two infrastructure types: an 802.11 wireless LAN and a wired network. Three kinds of devices are used to access content: a personal device assistant (iPAQ 3600) running under Windows CE and connected through the wireless network, a laptop computer using both the wired connection and the wireless one and a personal computer that uses the wired network.

In the proxy-based architecture a negotiation module is added to the communication proxy in order to retrieve and request the different client profiles. The same module is integrated to the content server in the server-based architecture.

The difference between the two architectures is that in the case where we use an intermediate proxy, we can not experiment all the aspects of the content negotiation and adaptation strategy. In this case, tasks like evaluating the available content variants and content substitutions can not be performed correctly due to the lack of control on the content.

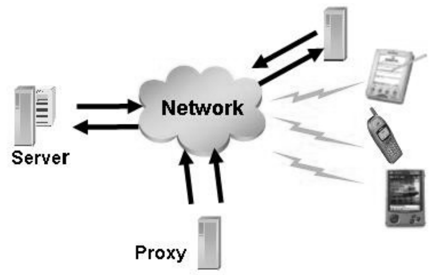

Figure 6. Server and proxy based Architecture

The proxy-based architecture is a very suitable for the context change support. In this case, the proxy is the entity responsible of retrieving client contexts and looking for the eventual changes that may occur. These changes are therefore transparent for the content server. The proxy can transform existing multimedia content and thus the content server is not directly involved in the adaptation.

At this stage, the client profile is always stored in the same location as the user context module. At any moment, the profile can be changed by selecting another profile.

6. Dynamic transformation and adaptation

An adaptation method *M* is applied on a resource *R*, if the UPS description of *R* matches the UPS input requirement of *M*, and the UPS output description of *M* matches the client requirements. To avoid developing static adaptation methods for each kind of resources adaptation, it's preferable that the server has methods that provedes many outputs depending to the context of their application. This is called dynamic adaptation. A dynamic adaptation is the one that interacts with the current client context. Taking into account the client context by an adaptation method can be achieved at two locations: the client or the server (or proxy).

In the general case, the client content adaptation is achieved using scripts and rendering styles which are sent inside the content and evaluated during the rendering according to the capability of the user device. Unfortunately, this kind of adaptation has many disadvantages and depends widely on the client processing power.

Dynamic adaptation on the server represents a best alternative to deal with the variety of client contexts. It allows the delivery of adapted content directly. In our system, we consider two kinds of adaptations: the structural adaptation (transformation) and the media adaptation. Structural transformations are based on XSLT [19] which allows transforming XML document into other XML documents. Media adaptations use specific applications.

6.1. Using the XSLT language

Providing XSLT style sheets that handles client profiles is seems interesting since the server can obtain adapted content by applying the generated style sheet to content. It is also possible to concatenate the original service with the user agent context (or profile) and then apply the style sheet. In practice, this is very complicated to achieve using XSLT templates. The two parts of the input tree (the constraints profile and the original content, see Figure 7) are separated and requires intensive processing to achieve the cooperation between them.

The scheme that we propose here consists to define a generic style sheet that admits as input the client profile and generates as output a style sheet.

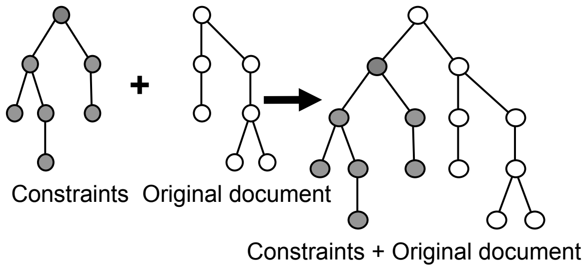

Figure 7. Support of user constraints by concatenation.

So, the generation of the style sheet that will be used in the final adaptation is done automatically. This adaptation will satisfy the received set of client constraints and is applicable for all documents with the same constraints.

In our proposed solution, the time of the adaptation of *n* content instances equals to the creation time of the generated style sheet plus the time of the adaptation of the *n* instances. This time is very smaller if compared to the one required in the first discussed solution. In this last, the corresponding time equals to: the time of the concatenation and the creation of the original content with the client profile, plus the time required for adapting the output document. Formally, the adaptation time *T* of *n* content instances equals to:

a) *In the first solution*:

T = *n* . time (concatenation_And_Creation (Client_Profile, Original_Content)) + *n* . time (adapting (Created_Document))

b*) In the second solution*:

T = 1 . time (creation_of_the_Generated_Style_Sheet (Client_Profile, Predefined_Style_Sheet)) + *n* . time (adapting (Original_Content)).

The creation of the generated Style Sheet is done once for the same client profile.

In the following, we give an application example of our approach. The example presents a dynamic filtering of SMIL [18] content. The figure 8 gives a simple client profile that indicates the non support of audio and parallel scenarios executions. Figure 9 represents the generic XSLT style sheet used for filtering. The style sheet of the figure 10 is generated automatically following the proposed approach and it can be used to transform SMIL content according to the user constraints given in the client profile.

```
<ClientProfile>
<Service category="smil" name="Synchronized Multimedia
Integration Language">
  <SoftwarUsed>RealPlayer 8 Basic 6.0</SoftwarUsed>
  <ServiceComponent category="video" tagName="video"
support="yes">
    <SourceType>
      <NotSupportedBag>
       <li>mpeg</li>
      </NotSupportedBag>
    <OnlySupportedBag>
    </OnlySupportedBag>
    </SourceType>
</ServiceComponent>
<ServiceComponent category="image" tagName="img"
support="yes">
   <MaxDisplay>100x200</MaxDisplay>
   <MaxSize>2000K</MaxSize>
</ServiceComponent>
<!-- support=no, the service is completely not
supported -->
<ServiceComponent category="audio" tagName="audio"
support="no">
</ServiceComponent>
<!-- support=noTag, the sevice tag is not supported,
but its content is -->
<ServiceComponent category="paralel execution"
tagName="par" support="noTag">
</ServiceComponent>
</Service>
</ClientProfile>
```

Figure 8. The client profile

```
...
<xsl:template match="ServiceComponent">
<xsl:variable name="E">
<xsl:value-of select="@tagName" />
</xsl:variable>
<xsl:choose>
 <xsl:when test="@support = 'no'">
<!-- Non supporting services processing -->
<xsl:text>
</xsl:text>
<xsl:element name="xsl:template">
<xsl:attribute name="match">
<xsl:value-of select="$E"/></xsl:attribute>
<xsl:text>
</xsl:text>
</xsl:element>
</xsl:when>
<xsl:when test="@support = 'noTag'">
<!-- Non supporting services tag processing  -->
<xsl:text>
</xsl:text>
<xsl:element name="xsl:template">
<xsl:attribute name="match">
<xsl:value-of select="$E"/></xsl:attribute>
<xsl:text>
</xsl:text>
<xsl:element name="xsl:apply-templates"></xsl:element>
<xsl:text>
</xsl:text>
</xsl:element>
</xsl:when>
<!-- End of the non supporting services processing -->
<xsl:otherwise></xsl:otherwise>
</xsl:choose>
</xsl:template>
...
```

Figure 9. The generic style sheet for filtering

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output omit-xml-declaration="yes"/>
<xsl:template match="*">
<xsl:copy>
<xsl:apply-templates select="@*"/>
<xsl:apply-templates/>
</xsl:copy>
</xsl:template>
<xsl:template match="@*">
<xsl:copy/>
</xsl:template>
<xsl:template match="audio">
</xsl:template>
<xsl:template match="par">
<xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

Figure 10. The generated XSLT style sheet

Unfortunately, XSLT transformation is applied at the document structure level and can't ensure advanced adaptations such those depending to the client screen, color or resolution supports, etc. Consequently XSLT transformations must be enriched by other kind of adaptations that acts directly on media elements.

6.2. Using media adaptation methods

This kind of adaptation concerns media resources such as images, videos, etc. A context is considered as a set of variables that reflect the state of the client capabilities and preferences. Consequently, a context change is equivalent to a values change of these parameters. The new values of the context are taken as input and the corresponding adaptation methods are applied.

Here are some experimental results concerning the dynamic adaptation used within the exchange protocol. In each application, a user is associated with as a set of variables used in the adaptation.

Application 1: Image generation using structural transformation (Figure 11): XML to SVG [17]. The client context about the requested content includes the following set: {wanted_text="Structural Transformation", image_width = 400, image_height = 60, preferred_font_color = "blue", preferred_font_size = 30, preferred background color = "red", preferred text position=40}.

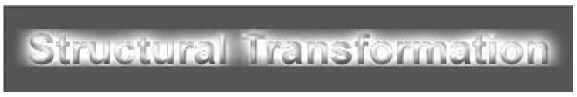

Figure 11. Created Image

Application 2: Images compression (Figure 12). The client context includes the following set: {Jpeg_Image_compression = 90%}.

Application 3: Images resizing (Figure 13). The client (PDA) context includes: {screen_width = 240, screen_height = 320}.

Figure 12.a: Requested content

Figure 12.b Image compression

Figure 13.a: Requested content

Figure 13.b: Adapted content (Resizing for 240X320)

7. Conclusions

In the current internet, providing adaptable content delivery is crucial. This becomes even more important when considering multimedia content which requires the handling of resource intensive media.

As stated in the paper, defining a complete adaptable content delivery system is not an easy task. It introduces many challenges at different levels of the current infrastructure. One of such challenges is the definition of a model for describing the environment or the environment constraints that have to be taken into account. UPS was proposed as a flexible model for describing not only the client but also the content and the server or the proxy capabilities.

We have presented also a protocol and a negotiation and the adaptation strategy which allows the delivery of the final content to the user agent. In order to support the dynamic context changes, two principles were presented concerning structural transformation -using XSLT- and media adaptation using direct transcoding methods.

The defined framework remains extensible especially for the context description. This allows to handle new terminals type in the proposed framework.

8. References

[1] Chi Chi Hung and Lim Yan Hong. Adaptive Proxy-based Content Transformation Framework for the World Wide Web. IEEE 2000.

[2] Christianson L. and Brown K. Rate Adaptation for Improved Audio Quality in Wireless Networks. The 6th IEEE International Workshop on Mobile Multimedia Communications MOMUC'99, San Diego, California, USA, 15 - 17 November 1999.

[3] Dey A. K. and Abowd G. D. Toward a Better Understanding of Context and Context –Awareness. GVU Technical Report, 1999.

[4] Fielding R. and al. Hypertext Transfer Protocol - HTTP/1.1. Internet Draft, November 18, 1998.

[5] Frystyk N. H., Leach P. and Scott L. HTTP Extension Framework. Internet Draft, http://www.w3.org/Protocols/HTTP/ietf-http-ext/draft-frystyk-http-extensions-03.txt, March 15, 1999.

[6] Graham Klyne, Franklin Reynolds, Chris Woodrow, and Hidetaka Ohto, Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies, http://www.w3.org/TR/CCPP-struct-vocab/, W3C Working Draft, 15 March 2001.

[7] Hidetaka O. and Johan H. CC/PP Exchange Protocol Based on HTTP Extension Framework. W3C Note, http://www.w3.org/TR/NOTE-CCPPexchange, 24 June 1999.

[8] Holtman K., TUE and Mutz A. Transparent Content Negotiation in HTTP. RFC 2295, Network Working Group, March 1998.

[9] John R. Smith. VideoZoom Spatio-temporal Video Browser. IEEE Trans. Multimedia, Vol. 1, No. 2, June '99.

[10] Lazar A.A. Challenges in Multimedia Networking. Proceedings of the International Hi-Tech Forum, Osaka `94, Osaka, Japan, February 24-25, 1994 pp. 24-33.

[11] Lemlouma T. and Layaïda N. Universal Profiling for Content Negotiation and Adaptation in Heterogeneous Environments. W3C Workshop on Delivery Context. W3C/INRIA Sophia-Antipolis, France, 4-5 March 2002.

[12] Network Working Group. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945: http://www.ietf.org/ rfc/rfc1945.txt, May 1996.

[13] Ora L. and Ralph S. Ressource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation: http://www.w3.org/TR/1999/REC-rdf-syntax.

[14] Rodden, T., Cheverst, K., Davies, K. Dix, A. Exploiting Context in HCI Design for Mobile Systems. Workshop on Human Computer Interaction with Mobile Devices, 1998.

[15] WebSphere. Transcoding Publisher. IBM Corp, http://www-4.ibm.com/software/webservers/transcoding/.

[16] W3C. CC/PP. http://www.w3.org/Mobile/CCPP/

[17] W3C. Scalable Vector Graphics (SVG). http://www.w3.org/Graphics/SVG/Overview.html8

[18] W3C. Synchronized Multimedia Integration Language (SMIL 2.0) Specification, http://www.w3.org/AudioVideo.

[19] W3C. XSL Transformations (XSLT) Version 1.0. W3C Recommendation. http://www.w3.org/TR/1999/REC-xslt-19991116.