Modeling and Design of an Adaptive Virtual Reality Service

Dario Mikic[1], Danko Vilendecic[1], Lea Skorin-Kapov[2], Maja Matijasevic[1]

[1]FER, University of Zagreb
Unska 3, HR-1000 Zagreb, CROATIA
{dario.mikic, danko.vilendecic, maja.matijasevic}@fer.hr

[2]Ericsson Nikola Tesla
Krapinska 45, HR-10000 Zagreb, CROATIA
lea.skorin-kapov@etk.ericsson.se

Abstract

This paper presents a model for an adaptive virtual reality (VR) service. The service adaptation, as described by the model, is performed by matching the parameters of – and possibly transcoding – the VR service in order to achieve the "best possible quality", given the limitations of the terminal and access network, and taking into account individual user preferences. The matching process is based on two sets of parameters, called "profiles": the VR service profile, which describes the VR service, and the client capabilities and preferences profile, which describes the client. Both profiles use Extensible Markup Language for notation, and are thus in line with the existing W3C standards. The model is independent of the particular VR service and network scenario, and it is suitable for heterogeneous environments such as the Internet and 3G wireless networks. A case study involving two prototype VR services demonstrates the applicability of the proposed approach.

1. Introduction

The concept of the next generation network [1] is guided by a vision of a multiservice, multi-access network of networks, providing a number of advanced services "anywhere – anytime". With the ever increasing demand for new services and applications [2][3], a *virtual reality (VR) service* may be considered as a yet another advanced service, in the area of merging of multimedia computing and (tele)communication technologies [4]. A wide range of VR services with tremendous market potential may be foreseen, ranging from virtual shopping and entertainment to data visualization, simulation-based applications, and education/training. Recently, there is a growing interest in augmented reality (AR) [5]. With advances in capabilities and varieties of mobile terminals [6] comes a challenge to provide such services to mobile users [7][8].

In this work, we are particularly interested *in non-immersive VR* delivered over the World Wide Web [9]. Such virtual worlds may be experienced on any "terminal", for example, a multimedia desktop PC, a hand-held/palmtop PC, a multimedia-enhanced mobile phone, or a personal digital assistant (PDA). Clearly, these terminals differ in terms of processing, storage and display capacity, as well as networking capabilities. In such a heterogeneous environment, the goal of a VR service provider may not necessarily be to provide "the best" (meaning, the most expensive) service – which may be rendered useless by, for example, a lack of display capability – but to provide the "the best achievable" service instead. In this scenario, the user gets the best possible service, given terminal limitations, and the service provider still profits.

The issue of terminal capabilities is not the only one source of heterogeneity. When discussing multimedia and VR services, human factors greatly contribute to the overall quality of service as perceived by the user. Thus, when determining the "best possible" service, user preferences must be taken into account as well.

In this paper, we propose a VR service adaptation model [10] that addresses adaptation in response to both terminal heterogeneity and user preferences. The proposed model represents an extension of a general VR framework [11] and it builds on existing standards and techniques for multimedia (trans)coding, compression, and networking, as well as 3D graphics optimization, in order to achieve actual service adaptation. It is also suitable for service architectures for personal mobile communications [12]. Other work in this area addresses VR-related protocol enhancements [13], as well as QoS adaptation of particular VR services [14][15]. As opposed to these specific approaches, we focus on a more general method of specifying and matching the sets of VR service parameters ("profiles") that adequately describe a VR service.

This paper is organized as follows. The proposed service adaptation model is presented in Section 2. Section 3 describes the implementation of the model, and Section 4 shows some preliminary results, using two simple prototype VR services. Section 5 concludes the paper.

2. Proposed service adaptation model

The purpose of the model is to describe the process of providing "highest achievable" VR service quality, from the point of view of the user and the "VR service provider". A distinction is made between the quality of service *(QoS) at the user/application level*, and the *QoS at the communication level*. The model, shown in Fig. 1, consists of three components:

- Client
- Access Server
- Application Server

The service invocation works as follows. Upon user's request, the client contacts the *Access Server* (the term "client" here denotes a particular combination of terminal hardware, operating system, and client software application). The *Access Server* identifies the client characteristics and determines and negotiates QoS. It then communicates with the *Application Server*, which is responsible for retrieving the VR service from the *VR service repository* and transforming the service as needed.

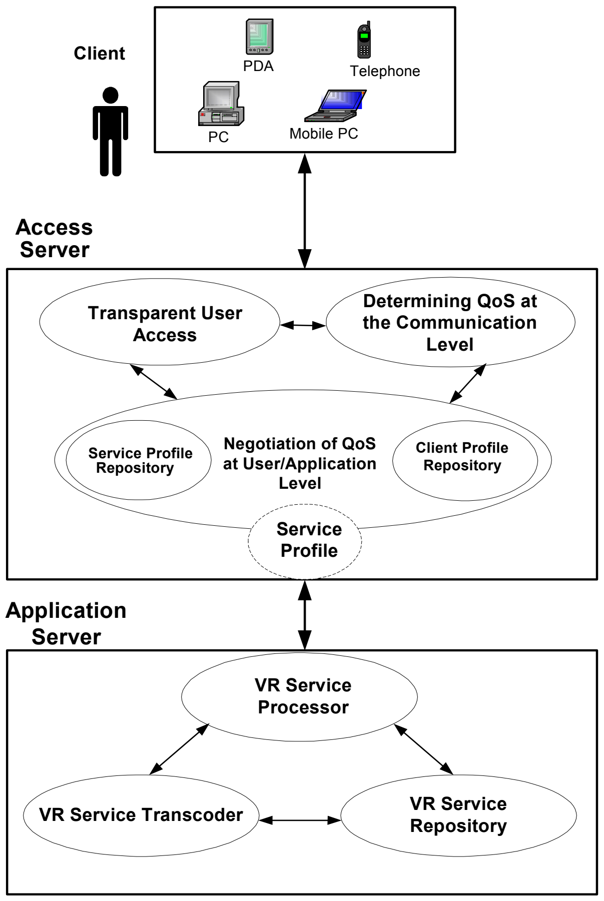

Figure 1. VR service adaptation model

Adapted content is returned to the *Access Server* and passed back to the client. It may be noted that the QoS negotiation and adaptation remains completely transparent to the client.

2.1. Access server

The *Access Server* is responsible for receiving the user request, identifying the user terminal, and determining the client profile. Based on the client profile, an appropriate service profile is then formed and passed to the Application Server.

The *Access Server* contains three modules providing the following functionality:

- Transparent user access
- Determining QoS at the communication level
- Negotiation of QoS at the user/application level

These modules are described in more detail in the sequel.

Transparent User Access

The role of the *Transparent User Access* module is to identify the access network and terminal of the user requesting the VR service. Two methods for implementing this function include:

- Identification based on the *User-agent* field in the HTTP header [16].
- Identification based on Composite Capabilities/ Preference Profile (CC/PP) [17]. CC/PP is a client profile data format used for describing device capabilities and user preferences based on the Resource Description Framework (RDF) [18].

After receiving the request and identifying the header fields, the *Transparent User Access* determines whether the request contains a CC/PP extension. If not, the client identification based on the *User-agent* is used. However, this approach lacks parameters needed for forming a complete client profile (parameters describing access network characteristics, precise terminal characteristics, and user preferences). Identification based solely on the *User-agent* field is therefore insufficient for supporting universal access to VR applications.

The preferred way of identification is via CC/PP profile, extending the HTTP request. The generic client profile (Fig. 2) is described using Extensible Markup Language (XML), and it consists of two parts: the terminal profile and user preferences profile. The terminal profile contains relevant and necessary information concerning the user terminal and access network (nominal, or otherwise known) communication capabilities. The user preferences profile describes parameters considered relevant by the user for achieving acceptable VR service quality. Acceptance parameters allow a user to choose the desired service format, along with whether or not to accept textures, audio or video. In terms of performance request parameters, a user may wish to specify a maximum acceptable download time. Performance optimization refers to whether the user wishes functional or visual service optimization to be pursued during the matching process. Special options are included for extending "standard" user preferences in special cases (such as availability of new services and technology).

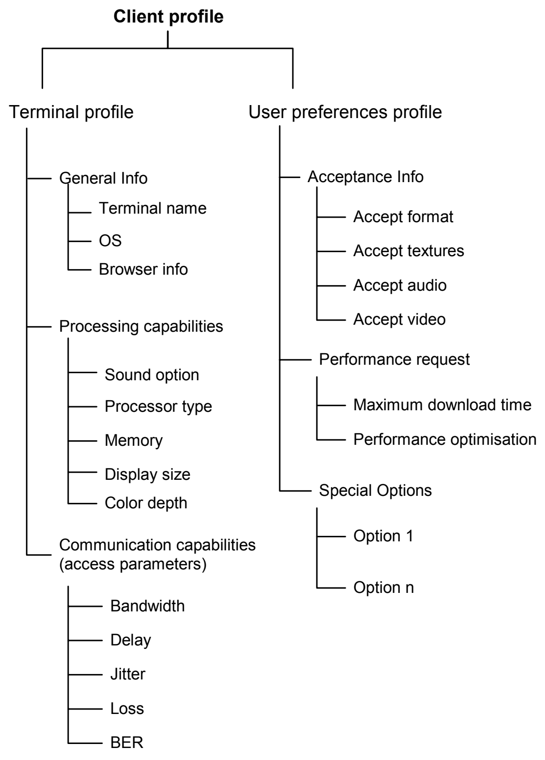

Figure 2. Generic client profile parameters

Determining QoS at the Communication level

The parameters passed on from the *transparent user access* module are important for determining QoS at the communication level. Again, we are considering the "best achievable" networking capabilities, and assume that further "downsizing" will follow as a consequence of networking conditions and QoS mechanisms. We plan to address this issue in more detail in future work.

The key network parameters for determining QoS for NVR services are available network bandwidth B and end-to-end latency L. We therefore define them as required parameters that are passed on to the module for *negotiation of QoS at the user/application level*. Jitter J and error rate E are passed on as optional parameters. Communication level QoS is defined as a function of the mentioned parameters:

$$QoS = f(\alpha \cdot B, \beta \cdot L, \gamma \cdot J, \delta \cdot E)$$

where $\alpha, \beta, \gamma, \delta$ are assigned a value of 0 or 1 to indicate which parameters are to be taken into account when adapting the VR service. For all classes of VR services, α and β are assigned a value of 1, while γ=1 for VR services using data streaming and δ=1 for VR services with a strict requirement on reliability. Example services and corresponding parameter values are given in Table 1.

Table 1. Example services and corresponding QoS parameter values

Example Service	α	β	γ	δ
VR with audio/video streaming	1	1	1	0
Virtual gallery (only download)	1	1	0	1
Collaborative virtual environment	1	1	1	1

The defined communication QoS level function is passed on to the next module to be used in the process of matching VR application QoS to access capabilities.

Negotiation of QoS at the User/Application Level

Based on information received from the user, the *Access Server* negotiates user/application level QoS in order to match service content to client capabilities and user preferences. Negotiation refers to the ability of the user to change his/her preferences with each request for a given VR service.

The module for *negotiation of QoS at the user/application level* receives and analyzes parameters from the previous two modules. This is followed by matching with VR application QoS (using the *service profile repository*) to define the final service profile of the VR service returned to the user.

Negotiation can be considered *passive* or *active*. Passive negotiation refers to the case when no user preferences are specified and client capabilities may only be guessed based on the value in the *User-agent* field. Based on this information, the format of the service (VRML, HTML, WML) that the client is capable of displaying is determined. Relevant profiles are retrieved from the *client profile repository* that match the user's browser information and contain default client parameters describing access networks and terminals. The unknown terminal characteristics are approximated by choosing the closest matching profile. Once the client profile has been determined, the corresponding default user profile is retrieved from the *service profile repository* and passed on to the *Application Server*.

Active negotiation is based on CC/PP specification. A well defined CC/PP profile contains all of the information necessary for precisely defining the client profile and user preferences. For example, a user may explicitly request that VR content be converted to HTML format. It should be noted that active negotiation may only be performed by a CC/PP-enhanced client. For example, WAP terminals do not support CC/PP specification and active negotiation.

VR Service Profile

Upon receiving the user's request for a particular service, the *service profile repository* offers all possible versions of that service. The "best" version (offering the highest quality as perceived by the user) is selected based on the terminal profile and is adapted according to user preferences. The final VR service profile enables the *Application Server* to choose the service content that best meets the agreed upon level of QoS. The *generic service profile* (Fig. 3) is described using XML and contains four sets of parameters: general service information, processing requirements, network requirements, and special options (used for parameters that need to be more precisely specified).

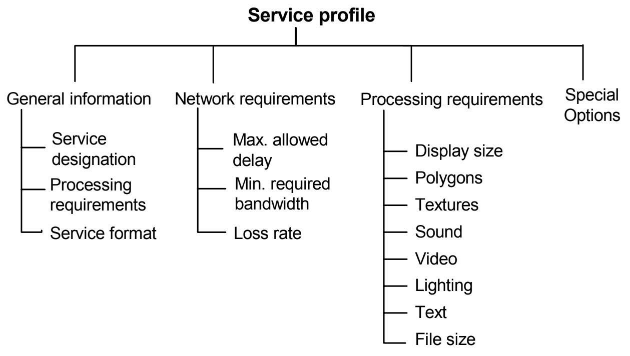

Figure 3. Generic service profile parameters

The agreed upon service profile is then passed on to the VR *Application Server* responsible for returning the desired content.

2. 2 Application server

The basic function of the *Application Server* is to deliver appropriate content based on the negotiated QoS level. Therefore it must have the capability of adapting VR content, whether using static or dynamic transcoding techniques. The *Application Server* consists of three separate modules:

- VR service repository
- VR service transcoder
- VR service processor

Depending on the physical location of these modules, the *Application Server* may be centralized or distributed. The core functionality of the *Application Server* is located in the *VR service processor*.

VR Service Repository

The *VR service repository* stores the files that make up virtual reality scenes offered by the *Application Server*. These files may contain:

- Source code of a virtual reality scene – in VRML [9], or XML format, according to X3D specification [19].
- Textures.
- Multimedia – sound files and video clips.

Depending on its relative position to the *Access Server*, the repository can be central, remote or distributed.

A central repository is positioned at the same physical location as the *Access Server*. This means that all the files that make up the VR service are stored on the same machine that contains the universal access logic.

A remote repository is positioned separately from the *Access Server*. One advantage of this approach is that it eases the processing load of the machine that executes the function of the *Access Server*, since the *VR service transcoder* is migrated together with the repository. Also, it can reduce the time necessary to transport the adapted content to the client, assuming that the repository is located "closer" to the client. "Closer" means that the repository is placed at a location to which the client has a faster connection. The disadvantage of this approach is the increase in time needed for *Access Server* to *Application Server* communication.

A distributed repository is a combination of the previous two approaches. The files that require longer downloads are migrated "closer" to the client, while the original service code is placed at the central location, together with the *transcoder* and the *Access Server.* The advantage gained by this approach is reduced processing and content transport time. The complexity of its maintenance is the main disadvantage of a distributed repository.

VR Service Transcoder

VR service transcoder is the *Application Server* module that carries out the transformation of the VR service format into a replacement format. The functionality of the transcoder is described by the following expression:

$$\mathrm{TR} = f\,(I, C, T)$$

where I is the set of input parameters provided by the service profile, C is the set of completed service contents stored in the repository and T is the set of content transformation functions. The set I consists of the following elements:

$$I \supset \{f, p, d, t, a, c\}$$

where f represents the format of the service, p the maximum supported processing speed, d the display size, t the texture quality, a the sound quality and c the number of sound channels.

Each of these elements is a set containing its own elements:

- $f = \{VRML, HTML, WML, null\}$
- $p = \{MIPS$ value$\}$
- $d = \{1280x1024, 1152x864, 1024x768, 800x600, 640x480, 480x240, 320x240, null\}$
- $t = \{16, 8, 4, null\}$
- $a = \{16, 8, null\}$
- $c = \{stereo, mono, null\}$

The elements of the set *C* are:

$C = \{vr_1, vr_2... html_1, html_2... wml_1, wml_2... xml_1, xml_2...\}$

where vr_n, $html_n$ and wml_n are the default service implementations in the appropriate format, while xml_n represents the universal service code written in XML.

VR service transcoder can may either use one of the "ready made" instances of the service, or, generate a service dynamically from the appropriate XML file using one of its transcoding functions:

$$T = \begin{Bmatrix} f(xml \to vrml), f(xml \to html), f(xml \to wml), \\ f(vrml_i \to vrml_j), f(html_i \to html_j), f(wml_i \to wml_j) \end{Bmatrix}$$

The transcoding process can be static or dynamic. The term *transcoding* is used here to denote the process of converting content from one format to another (e.g. from *VRML* to *HTML*) or the process of content modification (compression and filtering).

Static transcoding assumes the creation of different versions of the same content. The basic problem is in choosing the appropriate version of the content for a specific client using a specific terminal. The advantage of static transcoding is the reduction of processing time and load while executing the service. However, creating, organizing, testing and maintaining different versions of the service content is difficult and time consuming.

Dynamic transcoding separates the problem of content creation from the problem of creating different presentations. Dynamic transcoding is made up of a set of techniques for shaping the information that is to be delivered to the client. There are many different mechanisms of dynamic transcoding. For instance, the Extensible Stylesheet Language (XSL) may be used to convert the content stored as an XML file into a format appropriate for presentation on a specific terminal (*XSL Transformations* [20]). Other examples include converting HTML into other markup languages such as HDML, compact HTML and WML. C*lipper* mechanisms may be used to separate a subset of content, such that it can then be presented on a small-screen device. In VR applications the term *dynamic transcoding* describes the technique of using XML transformations to form a VR service from a basic XML file describing the service.

VR Service Processor

As mentioned earlier, the parameters of the VR service that is to be delivered to the client are defined in the service profile. The *VR service processor* receives and analyzes these parameters, adapts the content obtained from the repository to match the user defined QoS level, and delivers the adapted service or notifies the user that the service cannot be delivered due to limitations listed.

For instance, let us assume that a client requests a service using a hand-held computer with a GPRS connection. If the *Access Server* determines that the terminal has VR support, finds a matching profile, and returns the service as requested. Otherwise, the *Access Server* determines that the terminal has no VR support and creates the appropriate service profile. This profile specifies that the service should be delivered in HTML format. Upon receiving the service profile from the *Access Server*, the *VR service processor* obtains the service content from the *VR service repository* and sends it to the *VR service transcoder*. The *VR service transcoder* converts the service content from its original VRML version to an HTML version. The adapted content is then delivered to the client.

2.3 VR parameters needed for matching QoS

To precisely define and describe the VR service matching process, it is necessary to determine the fundamental parameters that affect its outcome.

QoS parameters

VR service QoS parameters determine the level of user satisfaction by affecting the way the service is shaped and transported over the network.

Parameters included in the VR service matching process are defined as elements of the set of negotiation parameters *P*:

$$P = \{p_1, p_2, ..., p_n\}$$

The set *P* must be finite and precisely defined so that the negotiation process can be executed relatively quickly (say, a few seconds) if not in real time.

For the implementation of the basic model for VR service adaptation, elements of the set *P* are defined as:

- p_1 – *service format*
- p_2 – *VR service display size*
- p_3 – *video content* (yes/no)
- p_4 – *audio content* (yes/no)
- p_5 – *textures*
- p_6 – *maximum download time*
- p_7 – *minimum bandwidth requirements*
- p_8 – *maximum delay permitted*
- p_9 – *maximum jitter permitted*
- p_{10} – *transmission loss*
- p_{11} – *visual quality* - determined by matching

processing capabilities, amount of memory and color depth of the terminal with the number of polygons, texture depth, lighting complexity, and application of textual representation of the service request.

- p_{12} – *audio quality* – denotes the matching of audio capabilities of the terminal with the audio requirements of the service.

All of these parameters, except for the parameter p_1, may have a NULL value, which indicates that the parameter should be ignored in the matching process. The matching of the values of individual parameters is done by the *Access Server*.

The UML sequence diagram shown in Figure 4 describes the basic functionality of the proposed model. At the top of the diagram we see the objects (classes, modules) involved in the matching process.

3. Model implementation

The proposed service adaptation model was implemented using the Java programming language (JDK 1.3.1). Java was selected because of its portability. This characteristic makes it easy to move individual components of the system, so that different topologies can be tested, for example the central and the remote repository.

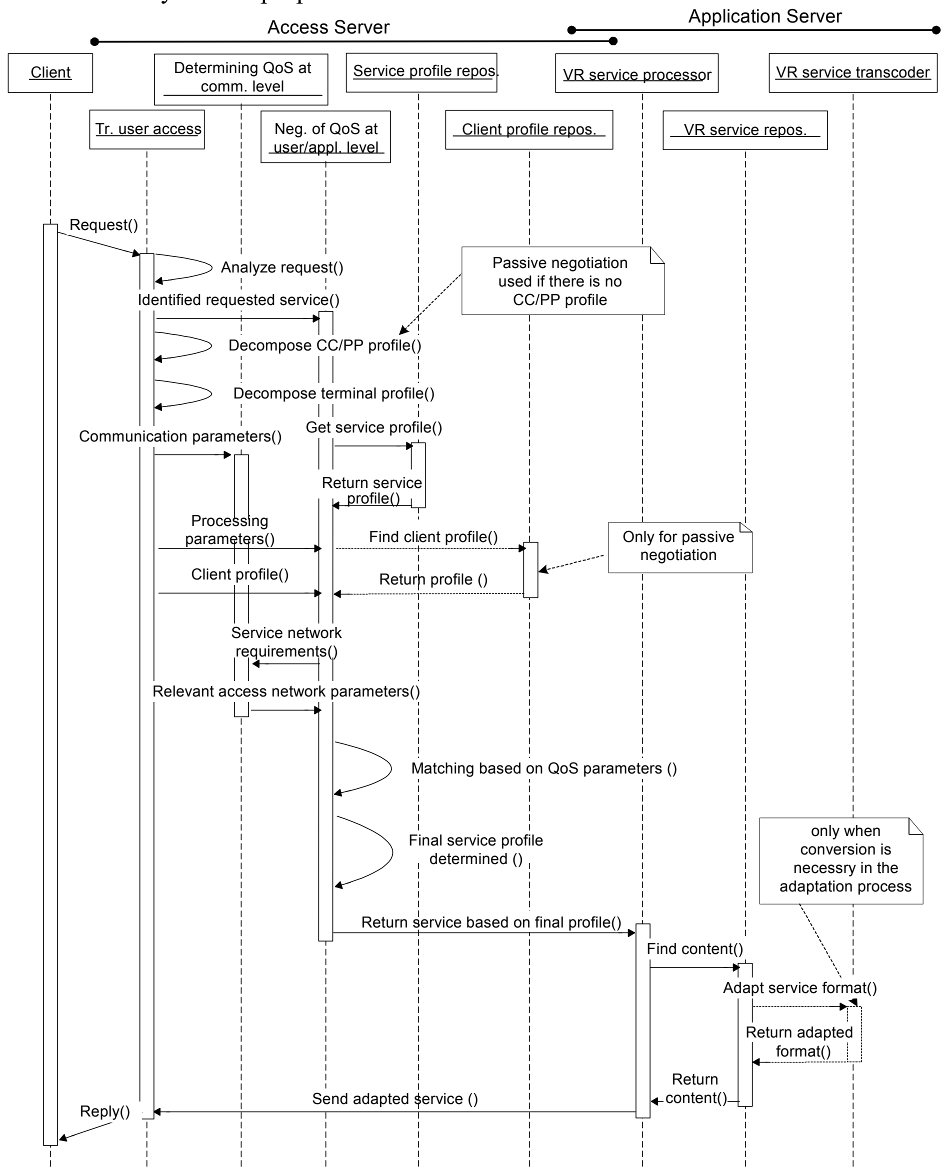
Access Server
Application Server
Client
Determining QoS at comm. level
Service profile repos.
VR service processor
VR service transcoder
Tr. user access
Neg. of QoS at user/appl. level
Client profile repos.
VR service repos.
Request()
Analyze request()
Passive negotiation used if there is no CC/PP profile
Identified requested service()
Decompose CC/PP profile()
Decompose terminal profile()
Get service profile()
Communication parameters()
Return service profile()
Processing parameters()
Find client profile()
Only for passive negotiation
Client profile()
Return profile ()
Service network requirements()
Relevant access network parameters()
Matching based on QoS parameters ()
Final service profile determined ()
only when conversion is necessry in the adaptation process
Return service based on final profile()
Find content()
Adapt service format()
Return adapted format()
Return content()
Send adapted service ()
Reply()

Figure 4. Sequence diagram for process of matching VR application QoS to access capabilities

As XML was used to describe client and service profiles, the integration is achieved through use of JDOM and XERCES APIs.

Following the structure of the proposed model (Fig. 1), each module, the Client, the Access Server, and the Application Server, was implemented as a separate Java class.

3.1. CC/PP capable client

As mentioned earlier in this paper, the proposed model implies sending CC/PP extended HTTP requests in order to achieve active QoS negotiation. Although the role of the CC/PP-capable client would normally be performed by the VRML-capable Web browser, as of April 2002, such browsers were not yet available. Thus, we developed a helper application which is capable of sending CC/PP extended HTTP requests.

This "new" client application enables a user to select a request containing the service identification, terminal profile and user preferences. An HTTP connection is then established to the *Access Server* and the request sent. The consequent response is displayed in a standard Web browser.

3.2. Access Server implementation

The functionality of the *Transparent User Access* module described by the model is implemented in the *TransUserAccess* class. This is the only module that the client side is aware of since all the communication with the client is carried out through it.

If the *Transparent User Access* module detects that a passive QoS negotiation method was requested (CC/PP profile is not present), service and client (read from the *User-agent* field of the HTTP request) identifiers are sent to the module for *negotiation of QoS at the user/application level*. Response received from the *Application Server* is forwarded back to the client.

If QoS negotiation is active, the *Transparent User Access* module decomposes the CC/PP profile and separates the terminal profile from user preferences. The terminal profile is additionally decomposed into processing capabilities and communication capabilities (Fig. 5). As already mentioned, CC/PP decomposition is done using JDOM and XERCES 1.4.3 APIs, which simplify parsing and manipulation of XML documents. Communication capabilities are sent to the module for *determining QoS parameters at the communication level*, while user preferences and processing capabilities are sent to the module for *negotiation of QoS at the user/application level*.

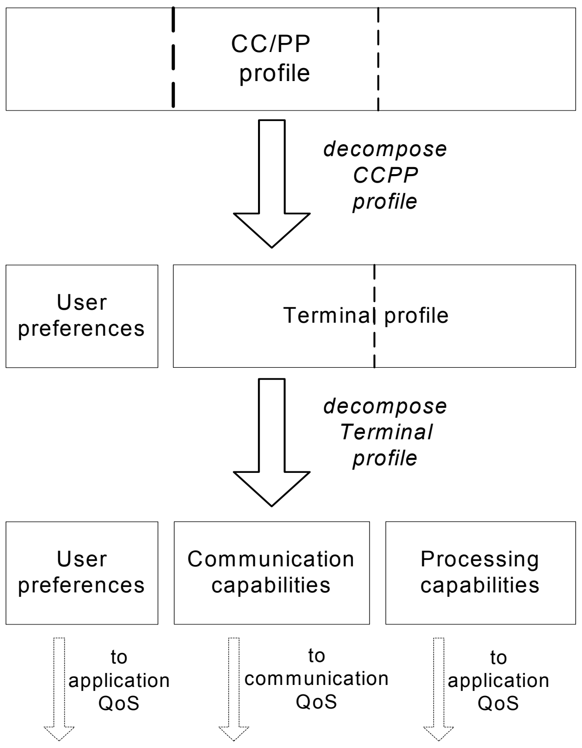

Figure 5. CC/PP profile decomposition

The *CommQoSDeterminator* class implements the module for *determining QoS at the communication level*. Its function is to compare the communication capabilities of the access terminal, which it receives from *Transparent User Access*, with the parameters defining the network requirements of the service, received from the module for *negotiation of QoS at the user/application level*. The result of the comparison are relevant network parameters to be used in the final process of matching according to defined QoS parameters.

The *AppQoSNegotiator* class implements negotiation of QoS at the user/application level. The matching of VR application QoS to access capabilities is done by comparing service QoS parameters provided by service profiles with the processing capabilities and user preferences received from the *Transparent User Access* module and relevant communication parameters received from the module for *determining QoS at the communication level*. The set of service profiles for the given service is retrieved from the *service profile repository* based on the service identifier.

If QoS negotiation is passive, the module for *negotiation of QoS at the user/application level* (*AppQoSNegotiator* class) uses the client identifier received from the *Transparent User Access* module to search the *client profile repository* for the default client profile corresponding to the access terminal. Matching with the service profiles in the case of passive negotiation is done only based on the *p1* parameter (service format).

If QoS negotiation is active, each parameter of each service profile is compared to the appropriate parameter defined in the client profile. If the client profile does not satisfy a requirement set by one of the service profile parameters, that service profile is excluded from the matching process.

The goal of the matching process is to determine the service profile that is to be sent to the *Application Server*. If more than one service profile matches the client profile, the highest quality service profile is sent.

The *service* and *client repositories* were implemented as part of the file system of the *Access Server*.

3.3.Application Server implementation

The *Application Server* was implemented using Apache 1.3.19 and Savant 3.0 Web servers, and Tomcat 3.2.1 Web application server. The logic of interaction with the *Access Server* was programmed in Java.

The *VR Service Processor* module was implemented as a Java application, which can be located at the same host running the Web or application server that is serving the application, or at a different host.

As previously mentioned, Apache and Savant Web servers were used to provide the functionality of the *VR Service Repository*. Since some services were located at a Web server running at a different host than the *Access Server*, the *VR Service Repository* can be classified as distributed.

The functionality of the *VR Service Transcoder* module was implemented by a Tomcat Web application server extended by Cocoon servlet technology. Cocoon was used to transform the service content from XML format to various presentation formats (HTML, WML, VRML).

4. Case studies

In order to verify the proposed service adaptation model and its implementation, two case studies were performed of two different VR applications: *Virtual Phone Gallery* and *Pyramid*. In both cases two different user access scenarios were tested using active QoS negotiation. These scenarios were defined by the CC/PP profile parameters (textures on/off, sound on/off, etc.) sent as part of the HTTP request. The testing environment for both case studies consisted of the following elements:

- A Pentium IV computer (1.6GHz, 512MB RAM) with Windows 2000 Professional operating system, running the client application.
- A Pentium IV computer (1.6GHz, 512MB RAM) with Windows 2000 Professional operating system, acting as the *Access Server*.
- A Pentium III computer (733MHz, 256MB RAM) with Windows 2000 Professional operating system, acting as the *Application Server*. Apache and Tomcat Web servers and the *VR Service Processor* module were installed on this host.

Virtual Phone Gallery application, developed in VRML, allows users to access a virtual shop containing virtual representations of several mobile phones. The service was implemented in two versions that differ in their complexity. The service profile belonging to the more complex version of the service identified it as having a display size of 1024x768 pixels, 10000 polygons, lighting complexity value 18, texture size of 298 kilobytes, texture color depth 24 bits, audio clip size of 32.5 kilobytes and file size of 1222.5 kilobytes. The simpler version had the same display size, 1970 polygons, lighting complexity value 1, texture size of 26.7 kilobytes, texture color depth 24 bits, file size of 255 kilobytes and no audio.

The case study for this application was conducted by using the client application (Fig. 6) to send two requests with different CC/PP profiles. The responses were then analyzed.

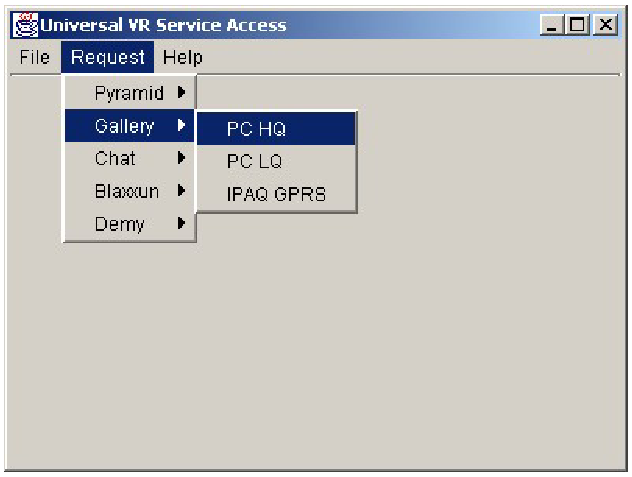

Figure 6. Client application GUI

The first request was sent using the "PC-HQ" (High Quality) user access scenario, which describes the access terminal as being a high-performance desktop computer. Since CC/PP profile parameters that define the "PC-HQ" access scenario satisfy the requirements, the more complex service version was returned to the client (Fig. 7 and 8).

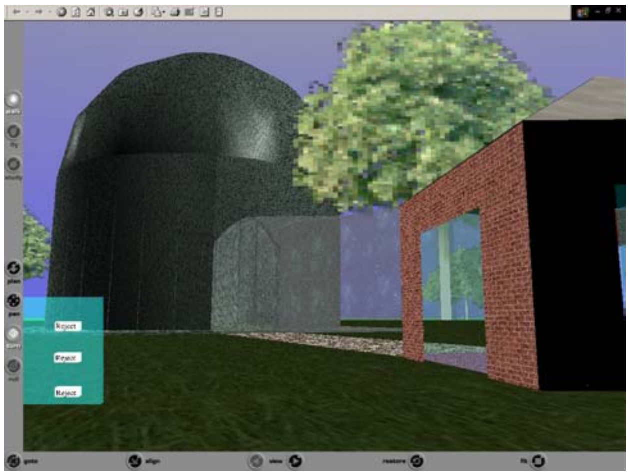

Figure 7. “PC-HQ” user access scenario response-outside view *(Gallery)*

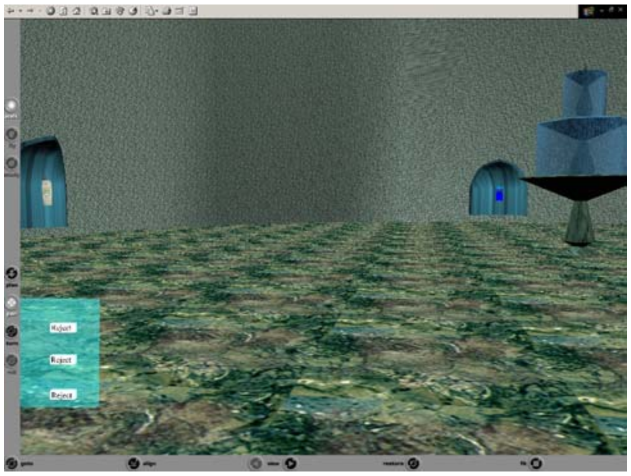

Figure 8. “PC-HQ” user access scenario response-inside view *(Gallery)*

The second request was sent using the “PC-LQ” (Low Quality) user access scenario, which describes the access terminal as being a “low performance” desktop computer. Since the CC/PP profile corresponding to that scenario does not satisfy the requirements of the more complex service version, the simpler version is returned (Fig. 9).

The deciding parameters in this case study were:

- the processor type and speed
- memory size and color depth of the access terminal
- number of polygons, lighting complexity, depth of textures of the service.

The *Pyramid* application, also developed in VRML, is a game of exploration in which a user’s goal is to find an object located in a hidden room inside the pyramid. Two versions of the service were implemented with the main difference being the display size.

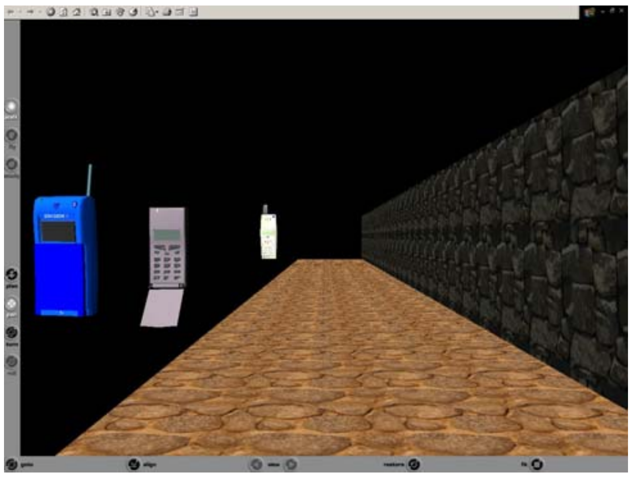

Figure 9. “PC-LQ” user access scenario response *(Gallery)*

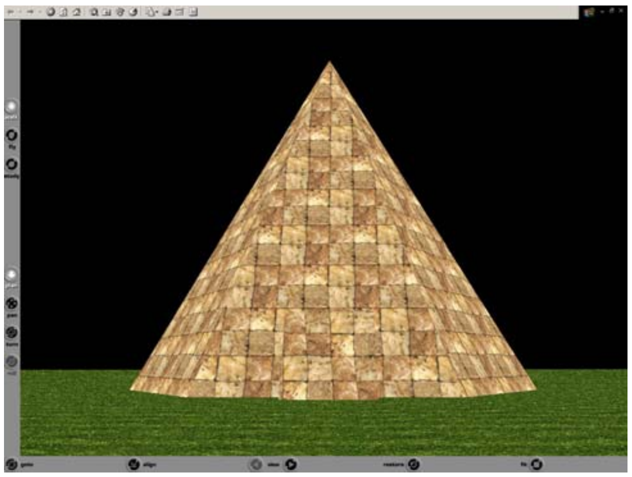

Figure 10. “PC-HQ” user access scenario response *(Pyramid)*

The case study for this application was performed using two user access scenarios: “PC-HQ” and “IPAQ-GPRS”. The “PC-HQ” scenario represented a “high performance” desktop computer, while the “IPAQ-GPRS” scenario represented an iPAQ pocket PC with a GPRS connection.

As in the first case study, the response received by the client (Fig. 10) showed that the CC/PP profile describing the “PC-HQ” user access scenario satisfied all the requirements of the more demanding service version.

The “IPAQ-GPRS” access scenario did not meet the requirements of the first service version, so the second version was returned instead. The deciding factor in this case was the display size. The service profile describing the first version defined the display size of the service as being 800x600 pixels, which exceeded the maximum display size of 240x320 pixels, as written in the CC/PP profile.

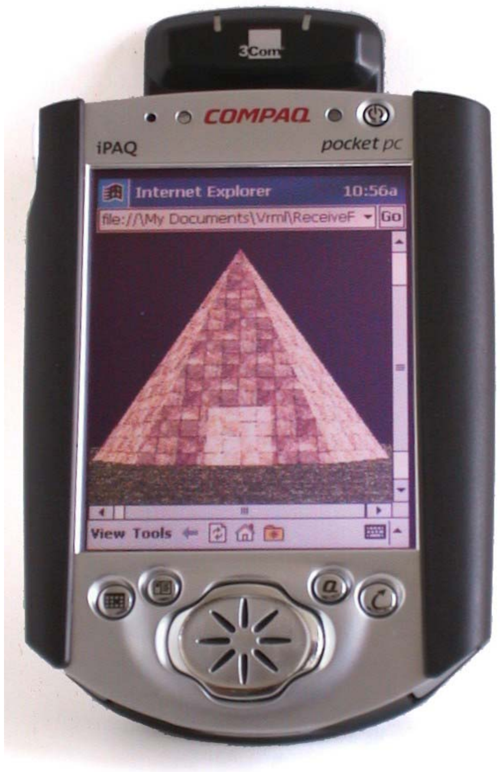

Figure 11. "IPAQ-GPRS" user access scenario response *(Pyramid)*

The display size of the second version of the service was 240x320 pixels, so the requirement was met. Figure 11 shows the response as viewed on an iPAQ pocket PC.

As demonstrated by case studies, the VR service was successfully adapted to terminal limitations as well as user preferences.

5. Conclusion and future work

In this paper, we presented a VR service adaptation model. The proposed model is independent of the underlying network and the VR service, which is its main advantage. Future work will address the effects of dynamic changes in profile parameters.

6. References

[1] A. Caric, K. Toivo. "New Generation Network and Software Design", IEEE Communications Magazine 38(2), February 2000, pp. 108–114.

[2] J.-H. Park, "Wireless Internet Access for Mobile Subscribers Based on the GPRS/UMTS Network", IEEE Communications Magazine 40(4), April 2002, pp. 38–49

[3] E. Ekkuden, U. Horn, M. Melander, J. Olin, "On-demand mobile media – A rich service experience for mobile users", Ericsson Review, No 4, 2001, pp. 168–177.

[4] M. Matijasevic, I. Lovrek, D. Mikic, A. Caric, D. Huljenic, "Designing Bandwidth-Aware Virtual Reality Services for the New Generation Networks", Proc. 9th Int. Conf. on Telecommunication Systems, Modeling and Analysis, Dallas, TX, March 2001, pp. 84-89,

[5] L. Rosenblum, "Virtual and Augmented Reality 2020", IEEE Computer Graphics and Applications 20(1), January/February 2000, pp. 38-39.

[6] T. Starner , "Thick Clients for Personal Wireless Devices", IEEE Computer 35(1), January 2002, pp. 133-135.

[7] S. Feiner, B. MacIntyre, T. Höllerer, and T. A. Webster, "A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment". In Proc. Int. Symp. on Wearable Computers ISWC'97, October 13-14, 1997, Cambridge, MA.

[8] C. Christopoulos, "Mobile Augmented Reality (MAR) and Virtual Reality", in *Book of Visions 2001*, Wireless World Research Forum, 2001, pp. 107–110

[9] D.R.Nadeau, "Building Virtual Worlds with VRML", IEEE Computer Graphics and Applications, March/April 1999, pp.18–29

[10] D. Mikic, *Matching virtual reality applications QoS to access capabilities*, M.S. Thesis, FER, University of Zagreb, 2002.

[11] M. Matijasevic, D. Gracanin, K.P. Valavanis, I. Lovrek, "A Framework for Multi-user Distributed Virtual Environments", IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, 32(4), August 2002.

[12] T. Kanter, "An open service architecture for adaptive personal mobile communication", IEEE Personal Communications 8(6), December 2001, pp. 8-17

[13] D. Brutzman, M. Zyda, K. Watsen, and M. Macedonia, "Virtual Reality Transfer Protocol (vrtp) Design Rationale", In proc. of the WET ICE 1997, Cambridge, Massachusetts, June 1997. pp. 179-186

[14] O. Seiwong, et al. "A Dynamic QoS Adaptation Mechanism for Networked Virtual Reality", Proc. of the 5th IFIP Intl. Workshop on QoS, May 1997

[15] Y. Ishibashi, S. Tasaka, and T. Iwama, "Adaptive QoS Control for Video and Voice Traffic in Networked Virtual Environments", Proc. of the 9th Intl Conf. on Computer Communications and Networks, Las Vegas, Nevada, 2000, pp. 638–642

[16] –, *Hypertext Transfer Protocol* -- HTTP/1.1, IETF RFC 2616, June 1999.

[17] –, *Composite Capabilities / Preference Profiles: Requirements and Architecture*, W3C Working Draft, July 2000 [On-line: http://www.w3.org/TR/CCPP-ra/]

[18] O. Lassila, R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, February, 1999. [On-line: http://www.w3c.org/TR/REC-rdf-syntax/]

[19] –, Information technology -- Computer graphics and image processing – eXtensible 3D (X3D), ISO/IEC draft specification, Web3D Consortium, February 2002 [On-line: http://www.web3d.org/]

[20] –, *The Extensible Stylesheet Language Transformations (XSLT)*, Version 1.1, W3C Working Draft 24 August 2001, On-line reference, http://www.w3.org/TR/xslt11/