A Framework for Dynamically Configurable and Reconfigurable Network-based Multimedia Adaptations

Oussama Layaida, Slim Benatallah, Daniel Hagimont
SARDES Project- LSR-IMAG-INRIA Rhône Alpes
Montbonnot Saint-Ismier Cedex
France
{Oussama.Layaïda, Slim.Benatallah, Daniel.Hagimont}@inrialpes.fr

Abstract

The recent advances in mobile equipments and wireless communications are making the Internet more and more heterogeneous. Terminals vary in their network capacities (bandwidth, latency, etc.), hardware capacities (CPU, memory, screen size, etc.) and in their software capacities (data formats, communications protocols, etc.). On the other hand, this evolution adds to the dynamic nature of the Internet, where resources vary unexpectedly. Thus, providing an efficient access to multimedia services requires that multimedia data be adapted according to each terminal capacity. One approach to this issue is based on the use of intermediate nodes to perform such adaptations (media transformations, protocol conversion, and data transcoding). In this area, there has been much research work for tackling the various forms of heterogeneity. However, less attention has been paid to the dynamic configuration and the reconfiguration at run-time of such mechanisms. This paper addresses these two aspects and describes a framework for network-based adaptations in multimedia streaming applications. This framework relies on a language, named APSL (Adaptation Proxy Specification Language), allowing the specification of network-based adaptation processes and their run-time behavior. The translation of this language is based on a component-based platform offering basic multimedia-related functionalities; composed dynamically in order to build an arbitrary configuration. Also, reconfigurations are facilitated through high-level component-related operations. This paper describes the architecture of the framework and shows through experimental evaluation the benefits of run-time reconfiguration.

Keywords: Multimedia Adaptation, Proxy, Middleware, XML, dynamic reconfiguration.

1 Introduction

The last decade have seen the emergence of a wide range of networked devices such as, Personal Digital Assistant (PDA), Smart Phones, eBooks, etc. Such devices have the ability to handle various data types including text, image, audio and video. This evolution opens promising perspectives in the development of Internet services. Indeed, multimedia services generally used on desktop computers such as audio and videoconferencing, IP-telephony and video on demand, may be integrated on these equipments. However, this challenge is today faced to the heterogeneity of Internet. This heterogeneity appears in (i) network links (802.11, Ethernet, ADSL, etc.) that may differ in bandwidth, latencies, etc. (ii) the hardware capacities of terminals in terms of CPU, memory, screen size, battery life time, etc. and (iii) the software capacities of applications that may use different data formats (MPEG, H.26x, MJPEG, etc.) or different protocols (H.323, SIP, etc.).

To solve this problem, it becomes necessary to provide adaptation techniques that help applications to take into account the characteristics of their environment. This need of adaptation is especially important for distributed multimedia applications, which are characterized by an intensive use of host and network resources, and require time bounded processing and communications. Furthermore, multimedia applications need to change their behavior at runtime to fit the conditions of their environment at any time. Effectively, due to the variation of resource availability such as network bandwidth, CPU load or battery life time, multimedia applications have to perform reconfiguration tasks that adapt their initial configuration according to the new constraints. Atthe same time, different multimedia applications may have different reconfiguration requirements. For example, a real-time videoconferencing application tolerates low video quality when network congestion occurs, while a VoD application (Video on Demand) would prefer to apply a FEC algorithm (Forward Error Correction) as it is streaming a stored video file. This difference of reconfiguration need may also happen for heterogeneous terminals within the same application. For example, the videoconferencing application may need different reconfiguration policies for

a PDA than for a workstation, as the PDA has limited hardware capacities.

This paper proposes to address these issues using a flexible framework for building network-based multimedia adaptations. This framework combines the use of declarative languages and component-based technologies in order to deal with the broad range of requirements coming from the network, hardware and software capabilities of terminals.

The rest of this paper is organized in 5 sections. The next section gives an overview of related work. Section 3 introduces the adaptation framework and its design concepts. Section 4 is devoted to the description of APSL (*Adaptation Proxy Specification Language*), a language for the specification of proxy-based adaptation. Thereafter, section 5 presents the component-based multimedia framework. Then, section 6 describes the experiments and the obtained results. Finally, we conclude and describe our future work.

2 Related Work

The field of adaptive multimedia streaming has been actively explored during last years. The reader may find a survey and a comparison of existing approaches in [23]. These approaches can be broadly classified into: sender-based, receiver-based and network-based adaptations. In the sender-based approach, the sender regulates its transmission according to network conditions [20]. The adaptation mechanism consists in tuning key encoding properties (quality factor, frame rate, etc.) in order to reach the desired transmission rate. This technique allows a better utilization of network resources with a single transmission. However, it is limited to the adaptation of the transmission rate and does not scale to multiple heterogeneous receivers. The second adaptation approach left adaptation decisions to receivers. The suppliers of multimedia content provide several versions or multiple cumulative layers [13] corresponding to different levels of QoS. The receiver can then select the content, or the part of content, that agrees with its capabilities. While this approach seems more appropriate, it does not scale to highly heterogeneous environments, such as Internet. Considering the high number of QoS requirements, it requires the management of several versions of the same content, one for each possible case. The network-based (or proxy-based) approach consists in integrating media adaptation at intermediate nodes in order to adapt the multimedia content according to network, hardware or software capacities [6]. The benefit of network-based adaptation is its non-intrusivity. Indeed, using infrastructural proxies, it is possible to generate the appropriate content for each receiver transparently to existing servers. These advantages have motivated numerous works; we give below an overview of the most significant of them.

The Video GateWay architecture, called VGW [1], addresses the problem of matching the transmission quality of video streams with the available bandwidth. In the experiment reported by authors, this proxy transcodes an original Motion Jpeg stream transmitted at a 1 Mbps into a 128 kbps H.261 stream, which is more appropriate for low bandwidth networks. The conversion is made in the DCT domain, which offers an optimized transcoding scheme. The gateway can be parameterized by an external control interface that allows specifying encoding parameters such as: frame rate, output rate, etc.

In [8], a filtering proxy is placed inside the network to reduce bandwidth requirements of MPEG video streams. The proxy monitors the network state and drops MPEG frames when congestion occurs. To minimize the impact of this operation on the temporal quality, the proxy preserves I-frames (encoded in Intra-mode without dependencies with other frames) and selects P and B-frames in the dropping operation. A similar work was conducted for H.263 for spatial resolution down-scaling of the video content [12]. The limitations of such techniques are two fold: (i) they focus only on adaptations for network capacities and (ii) adaptations techniques apply on a specific encoding format.

In [16], the adaptation architecture use a structure called InfoPyramid to represent media content at multiple modalities and resolutions. Media object can be transformed within the same media type or converted into other media types (audio into text or video into images). Media transformations are made offline and the various versions for the same content are stored on the media server, which delivers the appropriate one for each client. The drawback of this approach is that it requires analyzing the media content and the generation of multiple versions that fit every possible situation, which involves the use of high storage space for each server.

WebSmile [11] is an RTP to HTTP gateway that mediates between the WWW and the MBONE. WebSmile consists in a software component integrated within a web server connected into a multicast-capable network. It allows Web client to receive on their web browsers a video content originally transmitted in RTP multicast sessions on the MBONE (Multicast Backbone). Despite the protocol conversion, no transformations are applied on media and the client has to support the media content.

A centralized conferencing server is proposed in [21] to manage multiparty audio-conferences using the Session Initiation Protocol (SIP) [17]. The server acts as a central

control point for multiple peer-to-peer participants. In addition to sessions' control, it receives media streams from all participants and redistributes the appropriate stream back to each one. Optionally, the server may perform transcoding or mixing of audio streams if needed. The media capabilities of participants are conveyed using the Session Description Protocol (SDP) [7].

By making this review, we note that most of existing projects have focused on a particular problem and proposed codec-specific or protocol-specific solutions. On the other hand, they employ a static configuration and do not consider changes in the environment. Such solutions cannot apply to all encountered cases in the Internet, since multimedia applications use various protocols, encoding format and data properties.

3 Summary of contributions

To address the shortcomings of exiting works, our objective is to provide a flexible framework that: (i) address a wide range of adaptation requirements and (ii) provide customizable reconfiguration mechanisms to deal with dynamic changes and the specific needs of applications. The architecture of this framework is depicted in Figure 1.

The means allowing the customization of both adaptations and reconfigurations policies to application requirements is the use of a high-level specification language. In accordance with this, our framework offers the APSL specification language. APSL allows the description of an adaptation process as a graph of basic multimedia related tasks. Besides, it offers constructs to express reconfiguration policies that define how it should react to changes. Such specifications may easily be generated by an application level proxy (or server) (such as HTTP proxies, SIP conferencing servers, etc.), which controls multimedia sessions (establishment, changes and terminating) with participants, determines their requirements and sends adaptation requests as APSL specifications.

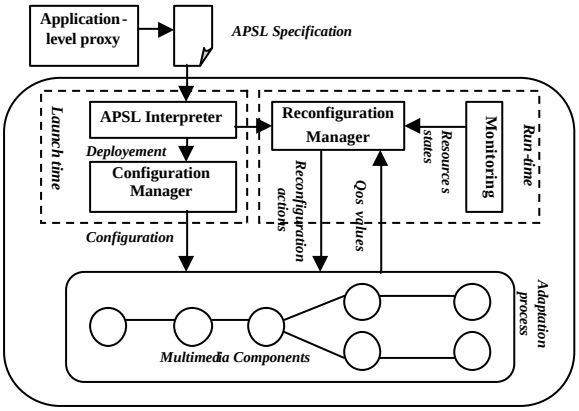

Figure 1 Framework architecture

At the top of the framework, an *APSL Interpreter* is in charge of mapping APSL specifications into run-time adaptation processes. A Configuration Manager offers the required functions in order to build the adaptation process as a composition of *Multimedia Components*. Each of these components implements basic multimedia-related task, which are composed to build the desired configuration. Once an adaptation process is launched, a *Reconfiguration Manager* is responsible for the execution of reconfiguration policies. As shown in Figure 1, we distinguish two kinds of events likely to activate reconfiguration actions: quality of service events (QoS) and resource states events. The former comprises events that may be generated by the adaptation process itself, such as packets loss rate, data processing rate, buffers' occupation, etc. The later represents changes in resource states such as CPU load, memory consumption, remaining battery life-time, etc. Such events are reported to the reconfiguration manager which decides when and how reconfigurations should be applied. Reconfigurations are performed at runtime by manipulating the initial configuration of the adaptation process. Using component-based implementation offers two kinds of reconfigurations:

- We call *Parameter reconfiguration* the modification of some properties of a component participating in the adaptation process. The adaptation process being built as a component-based architecture, it is possible to modify key properties of a given component in order to change the behavior of the adaptation process. For example, decreasing the encoding quality in order to reduce the transmission rate.
- *Structural reconfiguration* refers to the manipulation of the structure of the applications in terms of components and their relationships. For example, by dynamically adding new components in the configuration or removing existing ones.

Following sections detail the architecture of the framework. We illustrate through an application scenario the use of the APSL language and how dynamic configuration and reconfiguration are performed with the component-based platform.

4 The APSL Language

Let us consider an example of videoconferencing application with two heterogeneous terminals. A videoconferencing server distributes an RTP video stream encoded in MJPEG with a resolution 352*288. The first terminal accepts video in MPEG with a resolution 176*144 and uses the RTP protocol. The second terminal supports H.263 and UDP as transport protocol. To produce the

desired stream for each of them, a transcoding proxy mediates between the server and those terminals. The role of this proxy consists in adapting the original video stream for each terminal. Besides this operation, the proxy applies a loss-based congestion control algorithm to regulate the transmission of RTP streams. It monitors the packet loss ratio and adapts its transmission if significant changes are detected.

Figure 2 gives the APSL specification of the above scenario. As shown in this example, an APSL specification includes three main parts, defined as follows:

(1) The <NETWORK> part is dedicated to the specification of network media input and output streams. Each of them defines the network protocol, the address and the port number used for the stream.

(2) The <PROCESS> part specifies a functional description of the adaptation process applied on media streams. It consists of a set of typed functional elements, each describing a basic multimedia-related task such as, decoding, encoding, resizing, etc. Each of them has specific attributes that precise its behavior. For example, an encoder element has attributes for the declaration of its encoding format, frame-rate, quality factor, etc. Processing elements are bound using binding attributes (*ilink* and *olink*). The result of these bindings leads to a directed acyclic graph (DAG), representing the data processing sequence.

(3) The <RECONFIGURATION> part is used for the specification of reconfiguration policies. As shown in the example, it consists of one or more reconfiguration rules expressed in the form of observations, conditions and actions. An observation defines the *Events* likely to activate reconfiguration events. It may be a question of QoS-related observations (packet-loss, jitter, etc.) or resource-related events. Our example shows one QoS observation attached to the RTP transmission element with four events: (i) packet loss < 2 %, (ii) packet loss > 10 % (iii) packet loss > 50 % and (iv) Round Trip Time (RTT) > 200 milliseconds (ms). Reconfiguration actions are expressed as arithmetic operation on attributes' values (increase, decrease, set, divide, etc.). According to their purpose, we distinguish three kinds of actions:

- First, actions are applied on the functional attribute (e.g. of the process part). In the first condition of Figure 2, the action consists in increasing the encoding quality factor by 5 %. In the second condition, its value is decreased if the loss rate exceeds 10 %.
- Second, actions may manipulate the structure of the processing graph by modifying values of binding attributes (i.e. ilink and olink).
- Finally, actions may target reconfiguration policies themselves in order to change their behavior (observation thresholds, arithmetic operations, etc). As it is shown with the third condition, a first action consists in invalidating its states as the encoding format has been changed.

```
<APSL>
<NETWORK>
 <INPUT id="IN" src="rtp://194.199.25.52:5008"/>
 <OUTPUT id="OUT1" src="rtp://…." />
 <OUTPUT id="OUT2" src="utp://…" />
</NETWORK>
<PROCESS>
 <DECODER id="D" ilink="IN" fmt="26" />
 <RESIZER id="R" ilink="D" height="144"
  width="176" />
 <ENCODER id="E1" ilink="R" olink="OUT1"
  frame-rate="25" quality="100" fmt="32"/>
 <ENCODER id="E2" ilink="D" olink="OUT2"
  frame-rate="25" quality="80" fmt="34"/>
</PROCESS>
<RECONFIGURATION>
 <RULE id="R1">
  <PROBE element="OUT1" >
    <Event parameter="packet-loss" id="EVT1"
  op="falls" value="2" unit="%"/>
    <Event parameter="packet-loss" id="EVT2"
op="exceeds" value="10" unit="%"/>
    <Event parameter="packet-loss" id="EVT3"
op="exceeds" value="50" unit="%"/>
    <Event parameter="RTT" id="EVT4"
op=="exceeds"
value="200" unit="ms" />
 </PROBE>
 <CONDITION id="C1" events="EVT1" priority="1"
status="active">
  <ACTION name="increase" ref="id(E1)@quality"
value="2" unit="%"/>
 </CONDITION>
 <CONDITION id="C2" events="EVT2" priority="1"
status="active">
<ACTION name="decrease" ref="id(E1)@quality"
  value="5" unit="%" />
</CONDITION>
 <CONDITION id="C3" events="EVT3 AND EVT4"
  priority="2" status="active">
<ACTION name="set" ref="E1@fmt" value="31"/>
<ACTION name="set" ref="C3@status"
  value="passive"/>
<ACTION name="set" ref="C1@value" value="5"/>
</CONDITION>
 </RULE>
 </ RECONFIGURATION>
</APSL>
```

Figure 2 An APSL specification for video transcoding

5 Component-based Multimedia Framework

The previous section has given an overview of the APSL language. This section focuses on the component-based adaptation framework. It describes its architecture and shows its use with the APSL language.

5.1 Multimedia Components

The basic building blocks in the framework are the multimedia components, which are standalone software entities based on the DirectShow model [5]. A component encapsulates an atomic operation on multimedia data such

as, encoding, decoding, etc. It is characterized by one or more input stream interfaces used to receive data in a specific media type and one or more output stream interfaces for delivering transformed data in a specific data type. In addition, it may expose one or more configuration interfaces that allow setting-up its internal properties (buffer requirements, output rate, etc.).

In our framework, we distinguish two kinds of components: networking components and processing components. Networking components implement basic transport protocols (such as, TCP, UDP and RTP) and application-level protocols used within multimedia applications such as RTSP, HTTP, etc. A network receiver component receives data from the network and has only output stream interfaces to deliver data to processing components. In the same way, a network sender component receives data through its input interface and sends data into the network. Processing components, used between these two ends, integrates media-related functions for processing audio and video data. Examples of such components are decoders and encoders that convert data according to standard media formats such as MPEG-x, H.26x, MJPEG, etc. Other components offer spatial and temporal transformations of media data such as: frame dropping, resolution downscaling, color-space conversion, data insertion, media mixing, etc.

5.2 Configuration Management

The creation of an adaptation process is made by the translation of the APSL specification into a run-time configuration. As a first step, the APSL Interpreter checks the correctness of the specification. Besides syntactical issues, it checks the semantic correctness of the processing graph, in particular media-type compatibility between bound elements. After a successful parsing, the Configuration manager creates the functional part of the adaptation process. This operation is performed as follows:

- First, a composite component, named Media-Session, is created to own basic multimedia components.
- Second, for each functional element in the APSL specification, a set of multimedia components are created and configured with its attributes. The number of components created depends on the nature of the task described in the element. For example, the RTP output requires two multimedia components: one to build data packets with codec-specific RTP headers (not shown in figure 3) and another component to send data using RTP. At the same time, the UDP output will need only one component. During this process, elements' IDs, their functional attributes and their binding values are maintained within each component as they may be needed at runtime for reconfiguration tasks.
- Then, multimedia components are connected in the media session according to bindings in the APSL specification. Components are connected through their stream interfaces, where the output data type provided by a component matches the input data type of the component it is connected to.
- Finally, the execution of the resulting configuration is started.

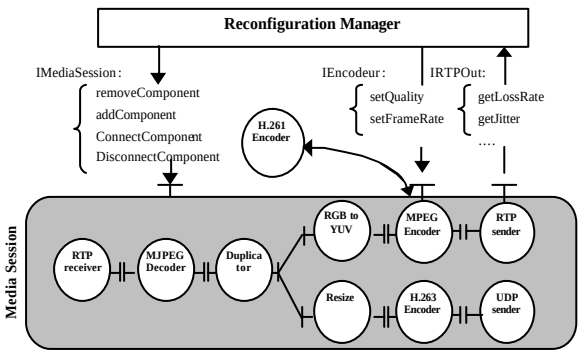

Figure 3 Configuration and reconfiguration of an adaptation processes

To clarify this process, we give in Figure 3 the Media-Session instantiated from the previous APSL specifications. It is composed of 9 media components. First, an RTP receiver and an MJPEG decoder are used to receive the original stream from the server and uncompress its content. Then, an MPEG encoder and a RTP sender produce an adapted stream for the first terminal and, an H.263 encoder and an UDP sender for the second. Notice that other components, not explicitly declared in the APSL specification, are inserted in order to perform non-functional tasks in the session. These components are used mainly to the coordination connections between functional components (e.g. networking and processing components). This is the case of the duplicator component on figure below, inserted between the MJPEG decoder and, the Resizer and H.263 encoder. As both elements have the same ilink value (D), a duplicator component is inserted after the decoder in order to duplicate its output into two streams. In the same manner, other processing components may be inserted for fine grain media conversion between components that accept strongly typed media formats. As the MJPEG decoder produces data in RGB and the H.263 restrict its input to YUV, an RGB-to-YUV converter is inserted to perform the required conversion.

5.3 Reconfiguration Management

During the translation step, the reconfiguration manager takes care of the creation of components for application monitoring and executing reconfiguration operations. In addition, it determines the media components involved by reconfigurations, that is: components that provide observations and components

Table 1 An example of parameter reconfiguration

```
monitoring_loop()
{
IRTPOut* RTPItf=
IRTPOut?getInterface(IID_RTPOut);
int loss=RTPItf?getLossRate();
int RTT=RTPItf?getRTT();
if(loss<2) notify(EVT1)
else if(loss>=10 && loss<50) notify(EVT2)
else if (loss>=50 && RTT>200) notify(EVT3) ;
}

notify (Event evt)
{
 if(evt==EVT1){ //EVT2 for condition 2
 IBaseFilter* encoder =
       Session.getComponent("E1");
 IEncoder* encItf=
      encoder?getInterface(IID\_Encoder);
 int quality = encItf?getQuality();
 quality+=2; // -=5 for condition 2
 encItf?setQuality(quality);
 } ….
}
```

upon which reconfigurations are applied. With a component-based platform, reconfiguration encompasses three basic tasks:

- The modifications of key component properties through their configuration interfaces.
- The insertion or/and the removal of components from the composite structure.
- The modification of bindings between components.

Complex reconfiguration operations can be divided into a set of those basic operations performed in an appropriate order. To clarify this concept, we consider reconfiguration examples given in our example (see Figure 2). We describe below how these operations are implemented using component interfaces (we focus on components manipulation). As shown on Figure 3, the entities involved by reconfiguration are: the RTP sender (OUT1), the encoder (E1) and the media session. The RTP sender plays the role of observer and provides the IRTPOut [*] interface to return statistics about the transmission. The reconfiguration manager uses this interface to periodically monitor the packet loss rate experienced during the RTP transmission as shown in table 1 (monitoring_loop function). This observation may lead to three reconfigurations, each requiring specific actions applied to the encoder and the session. The encoder component provides the IEncoder interface allowing the modification of encoding properties (frame-rate, quality, etc.). The media session provides the IMediaSession interface to manipulate its sub-components and their bindings.

Table 1 shows the code for the first reconfiguration actions: (1) the MPEG encoder is retrieved using its APSL

[*] In DirectShow/COM programming, a writing convention uses the prefix 'I' to indicate Interface names.

Table 2 An example of structural reconfiguration

```
if(evt==EVT3) {
 IBaseFilter* h261enc =
 SessionManager.createComponent("H261Encoder");
 Session?Pause();
 IBaseFilter* mpegenc =
    Session?getComponent("MPEGEncoder");
 mpegenc?Disconnect();
 Session?remove(mpegenc);
 h261enc = Session?addComponent(h261enc);
 h261enc?ConnectComponent(h261enc);
 IEncoder encItf=
 h261enc?getInterface(IID_Encoder);
 Session?Start();
 …..
}
```

'id' attribute[†] (2) its configuration interface (IEncoder) is retrieved by calling getInterface[‡] method and (3) the quality factor is increased by calling the setQuality method.

The third reconfiguration task requires: (1) the replacement of the MPEG encoder component by an H.261 encoder (2) setting the frame-rate at 20 frames-per-second. Table 2 shows how this can be performed using components' and session's interfaces. First, the H.261 encoder is created by the configuration manager. Then, the session is paused as its structure will be modified. The MPEG encoder is disconnected from other components and removed from the media session. The next operation consists in adding the H.261 encoder and connecting it to other components. The frame rate is set through the IEncoder interface of the H.261 encoder by calling the setFrameRate method. Finally, the session is restarted.

6 Experimental Evaluation

This section gives an experimental study of our framework. After the implementation details, we describe the hardware and software configuration of the experimental environment. Thereafter, we give a performance evaluation with several application scenarios.

6.1 Implementation

The framework has been entirely implemented in C++ under Microsoft Windows 2000 and using the Microsoft .Net 2003 development platform. The component-based environment has been built upon the Microsoft DirectShow Software Development Kit version 9.0 [5]. With its widespread distribution and its efficient implementation, this platform is now considered as a reference platform for building multimedia applications. Several application

[†] IBaseFilter is the basic interface for all DirectShow components

[‡] This method is provided by COM components to retrieve components' interfaces using their IDs.

Table 3 Structure of multimedia sessions

Scenario	1	2	3	4	5
APSL elements	2	4	4	9	11
Components	5	5	7	11	22

scenarios have been released and experimented using the framework. Examples are multi-party audio and videoconferencing servers, transcoding proxies, protocol translation gateways, etc. Many experiments were performed in an experimental heterogeneous architecture composed of workstations mobile PDAs. The obtained results have shown the benefits of adaptations on client performance, especially for terminals with limited resources [2] [26].

6.2 Evaluation Environment

In our evaluation, we used a Windows 2000-based PC with a Pentium 4 Processor at 2 GHz and 256 Mb of memory to play the role of an adaptation proxy (or server) with our framework. As client terminals, we have used mobile PDAs (Compaq IPaq H3970) equipped with an XScale Processor at 400 MHz and 64 Mb of memory, running PocketPC 2003 operating system. Those terminals are connected to a wired Local Area Network through a 2 Mbps 802.11b wireless network. They use different multimedia client application to access multimedia services. In particular, two multimedia services are provided to those terminals:

- A video on demand (VoD) server provides MPEG 1/2 movies with different resolutions and accessible via HTTP.
- A videoconferencing server distributes a real time H.261 video stream in CIF resolution and using an RTP multicast transmission.

6.3 Cost of Configuration

The first evaluation concerns the cost of configuration operations described in section 5.2. Our goal consists in evaluating the overhead introduced by the component-based implementation. The evaluation criteria are the deployment time and the CPU consumption at run-time. The deployment time is defined as the time made from the beginning of APSL parsing until the startup of the execution of the adaptation process. We made the measurements for the following application scenarios:

(1) An RTP gateway that receives a multicast stream from the videoconferencing server and sends the content to a unicast terminal.
(2) A protocol-conversion gateway that receives an HTTP video stream from a web server, transcodes its content from MPEG into H.261 and forwards the result using RTP.
(3) A video transcoding proxy performing a video transcoding from H.263 to H.261.
(4) The same transcoding proxy with two terminals (see Figure 3).
(5) A multi-user video conferencing server managing communications between 4 participants. It receives 4 video streams and produces a mixed video stream for each participant.

Table 3 gives for each configuration the number of APSL functional elements in the specification and the number of DirectShow components created for the session. The number of DirectShow components is higher than APSL elements as several DirectShow components may be created for each APSL element. In addition, as explained in section 5.2, there may be additional component for non-functional needs (as Duplicator) or fine grain transformation (as RGB-to-YUV).

Figure 4 gives the deployment time for each application. It varies between 30 ms for 2 components (scenario 1) to 110 ms with 22 (scenario 5). Regarding to scenarios 1, 2 and 3, the deployment time increases linearly with the number of multimedia components (with an average of 15 ms per component). Indeed, as multimedia components are hosted in dynamic link libraries, creating an instance of a component requires first loading its DLL. We note on the same figure that loading introduces a high overhead in the configuration process. At the other hand, the deployment time is attenuated in scenario 4 and 5 (with 5 ms per component). This is due to the fact that such applications include multiple instances of same components, which are instantiated from shared DLLs (e.g. the videoconferencing server creates RTP senders and receivers for each participant). In practice, several multimedia sessions are created at the same time which increases sharing DLLs.

Figure 5 gives the CPU consumption with each application scenario. It varies with the nature of the treatments performed in the media session. For the first application, tunneling RTP packets from a multicast connection to a unicast user does not requires high CPU use, which is kept under 3 %. The second configuration includes an MPEG decoding and an H.261 encoding operation, which gives an average CPU load at 5 %. The third configuration involves a real-time H.261-H.263 transcoding and transmission, which consumes 13 % of CPU. With two clients in configuration 4, the average CPU load reaches 18 %. Finally, the videoconferencing server uses 4 decoders and as much of encoders which leads to a CPU use of 28 %. These results are satisfactory as all applications were successfully built and executed.

6.4 Cost and Impact of Reconfiguration

The second evaluation concerns the reconfiguration

operations within multimedia sessions. The objective is to evaluate the cost of component replacement. We performed our experiment within the fourth configuration illustrated in Figure 3 when performing the reconfiguration process showed in Table 2. This operation consists in replacing an MPEG encoder component by an H.261 encoder when the packet loss rate exceeds 50 %. The evaluation criteria at the proxy side is the time needed for perform the previous reconfiguration task. The average time to perform this operation was about 43 milliseconds. As said before, the new component is created and configured before stopping the adaptation process. This task takes most of the reconfiguration time (˜ 28 ms) as it requires loading the component from a library. The application was paused during the remaining time (i.e. ˜ 15 ms) in order to replace the encoder component.

At the client side, we measured the performance of the application during the visualization of the video content. The goal here is to show the effect of the reconfiguration on the client performance. The quality metrics measured are: (i) the displayed frame rate which is the number of frames displayed per second and (ii) the packet loss rate observed by the application during the transmission. Results are reported on Figure 6 and 7. We distinguish on these figures three different states: (i) before (ii) during and (iii) after the reconfiguration process:

- Before the reconfiguration process, the packet loss rate observed by the client application was around 30 %. This rate increases until it exceeds 50 % and causes the reconfiguration (at the same time the proxy detects an RTT upper than 200 ms). The frame rate is under 20 frames per second and decreases when the loss rate increases.
- During the reconfiguration process, the adaptation process is paused on the proxy. Consequently, the frame rate decreases suddenly down to 0 fps, inducing a black-out time of 500 ms. Although the proxy has been paused during only 15 ms, this time is spent by the client application to detect the change of the encoding format (RTP payload and codec-specific header fields, see [18, 19]) and create a new decoder.
- After the reconfiguration has taken place, the client application accepts the new stream and starts displaying its content. The frame rate increases and reaches the original rate (25 fps). The loss rate is kept under 10 % as the new stream agrees with the client's capacities.

These results show that, besides content adaptation, reconfiguration improves significantly the performance of client terminals. At the same time, the cost of such operation is minimal knowing that the re-deployment of an equivalent configuration takes around 70 ms (scenario 3 on Figure 4).

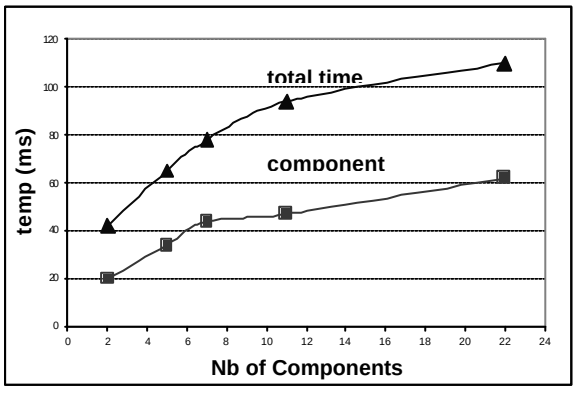

Figure 4 Configuration time for each application

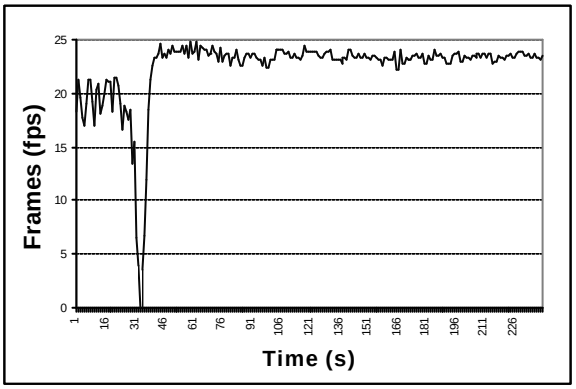

Figure 6 Displayed frame rate

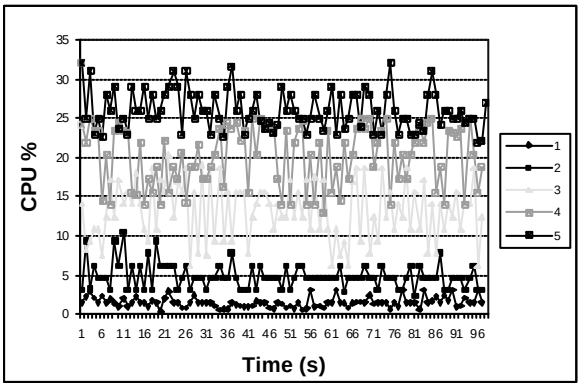

Figure 5 CPU load for each application

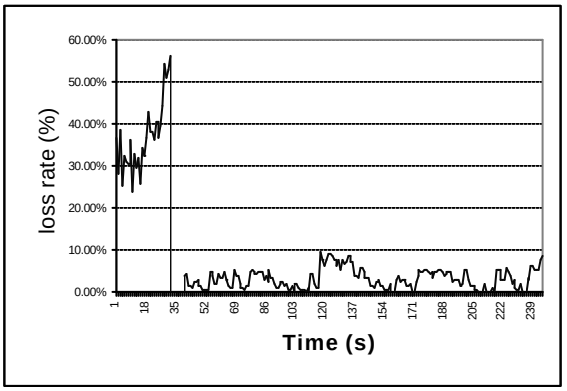

Figure 7 Packet loss rate

7 Conclusion and Future Work

This paper focuses on the dynamic configuration and the reconfiguration of media adaptation proxies. We have described a framework that offers flexible building of media adaptation services. Its component-based design allows the dynamically configuration of an adaptation process, and its reconfiguration during run-time. Besides, we proposed an XML-based language called APSL for Adaptation Proxy Specification Language. APSL is a high-level declarative language intended for the specification of adaptation process and reconfiguration policies to take into account changes in resource availability. Our future work concerns two aspects. First, we want to allow the remote configuration of a proxy node by submitting APSL specification. We are currently integrating a communication layer based on the Session Initiation Protocol [ROS02]. Our goal is to provide SIP/APSL-based gateways to automate the deployment of adaptation process. The second objective concerns the extension of the APSL language with the following aspects:

- The distribution of an adaptation process across multiple gateways distributed over the network. Effectively, for scalability issues, distributing adaptation on multiple gateways may be more interesting in many application scenarios.
- The integration of more advanced reconfiguration policies by considering new aspects such as fault tolerance, external reconfiguration events, etc.

Acknowledgements

This work has been partly funded by the IST Ozone project and by a Microsoft Research Innovation Excellence Award.

References

[1] E. Amir, S. McCanne and R. Katz. “An Active Service Framework and its Application to Real-time Multimedia Transcoding”. ACM Computer Communication Review, Vol. 28, No. 4, Sep. 1998, pp. 178--189.

[2] S. Ben Atallah, O. Layaida, N. De-Palma, D. Hagimont. “Dynamic Configuration of Multimedia Applications”. Proc of the 6th IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS'03), Belfast, Northern Ireland, September 6-10, 2003.

[3] G. Blair, L. Blair., V. Issarny, P. Tuma, A. Zarras. “The Role of Software Architecture in Constraining Adaptation in Component-Based Middleware Platforms”. Proc of Middleware 2000, LNCS 1795 - IFIP/ACM NY, USA, April 2000.

[4] T. Bray, J. Paoli, C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. Recommendation, W3C, February 1998.

[5] Microsoft DirectX (version 8.0): Microsoft DirectShow, Online Documentation: http://msdn.microsoft.com/directx/.

[6] A. Fox, S.D. Gribble, Y. Chawathe, and E.A. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. IEEE Personal Communications, 5(4):10-19, Aug 1998.

[7] M. Handley and V. Jacobson, SDP: Session Description Protocol. RFC-2327, Internet Engineering Task Force, April 1998.

[8] M. Hemi, U. Hengartner, P. Steenkiste and T. Gross. “MPEG system Streams in Best-Effort Networks”. Proc. of PacketVideo'99, Cagliari, Italy, April 1999.

[9] J. Indulska and al. “Experiences in using CC/PP in context-aware systems.” Proc. of IEEE Mobile Data Management 2003, Melbourne (2003).

[10] ITU-T Recommendation H.261: Video codec for audiovisual services at p x 64 kbit/s. Geneva, 1990, revised at Helsinki, Mar. 1993.

[11] M. Johanson. “An RTP to HTTP video gateway”. Proc. of the Tenth International World Wide Web Conference, Hong Kong , May 2001.

[12] Z. Lei and N.D. Georganas. “H.263 Video Transcoding for Spatial Resolution Downscaling”. Proc. of IEEE International Conference on Information Technology: Coding and Computing (ITCC) 2002, Las Vegas, Nevada, U.S.A, Apr. 2002.

[13] S. McCanne, V. Jacobson, M. Vetterli. ”Receiver-Driven Layered Multicast”. In SigComm'96, Stanford, CA, Aug 1996.

[14] S. McCanne, E. Brewer, R.Katz, L. Rowe, E. Amir, Y. Chawathe, A. Coopersmith, K. Mayer-Patel, S. Raman, A. Schuett, D. Simpson, A. Swan, T. Tung, D. Wu and B. Smith. “Towards a Common Infrastructure for Multimedia-Networking Middleware”. Proc. NOSSDAV'97, Missouri, US, 1997.

[15] S. Mitchell, H. Naguib, G. Coulouris and T. Kindberg. “A QoS Support Framework for Dynamically Reconfigurable Multimedia Applications”. Proc. of the 2nd IFIP International Working Conference on Distributed Applications and Interoperable Systems (DAIS '99), Helsinki, Finland, June 1999.

[16] R. Mohan, J.R. Smith, C. Li. “Adapting Multimedia Internet Content for Universal Access”. IEEE Transactions On Multimedia, Vol. l, No.1, March 1999.

[17] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[18] H. Schulzrinne et al. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, IETF, Jan. 1996.

[19] H. Schulzrinne. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 1890, IETF, Jan. 1996.

[20] D. Sisalem, H. Schulzrinne. "The loss-delay based adjustment algorithm: A TCP-friendly Adaption scheme". Proc. NOSSDAV'98, Jul 1998.

[21] K. Singh, G. Nair and H. Schulzrinne. "Centralized Conferencing using SIP". Proc. of the 2nd IP-Telephony Workshop (IPTel'2001), April 2001.

[22] J. Vass, S. Zhuang, J. Yao, and X. Zhuang. Efficient mobile video access in wireless environments. IEEE Wireless Communications and Networking Conference, New Orleans, LA, Sept. 21-24, 1999.

[23] X. Wang, H. Schulzerinne. "Comparison of Adaptive Internet Multimedia Applications". IEICE Transactions on Communications, June 1999.

[24]World Wide Web Consortium Working Draft, Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies, (http://www.w3.org/TR/CCPP-struct-vocab/), March 2001.

[25] World Wide Web Consortium Recommendation, XML Path Language (XPath) Version 1.0. Nov. 16, 1999. See http://www.w3.org/TR/xpath.html

Biographies

Oussama Layaida received his engineer diploma in computer sciences from the University of Sciences and Technologies of Houari Boumediene (USTHB, Algiers, Algeria) in June 2001, and his M.Sc. in Systems and Networking from the University of Joseph Fourier (UJF, Grenoble, France) in June 2002. Since, he is working toward a Ph.D. in Computer Sciences from the National Polytechnic Institute of Grenoble (INPG, Grenoble, France). He is actually a member of the SARDES Project at INRIA Rhône-Alpes research unit (Grenoble, France). His research interests include middleware technologies, adaptive multimedia systems and Quality of Service (QoS) provision.

Slim Ben Atallah is research scientist at INRIA Rhône-Alpes (Grenoble-France). He received his Ph.D. from University of Savoie in 1997. His was one of the main designers of OLAN system at INRIA-SIRAC Project, and he has worked in the area of distributed systems and groupware application support. In 1997-2002, he held an Assistant Professor position at "Ecole Nationale des Sciences de l'Informatique" (National High School of Computer Science) at University of Tunis, where he worked on system support for distributed collaborative applications. He is currently a member of the Sardes project at INRIA, Grenoble, where he is working on middleware support for adaptive distributed applications.

Daniel Hagimont is research scientist at INRIA Rhône-Alpes (Grenoble-France). He received his Ph.D. from Institut National Polytechnique de Grenoble in 1993. He was one of the main designers of the Guide distributed system at Bull-IMAG, and he has worked in the area of operating system support for distributed objects, distributed shared memory, and protection. In 1993-1994, he held a post-doctoral position at the University of British Columbia (Vancouver - Canada), where he worked on system support for distributed object-based languages and environments. He is currently a member of the Sardes project at INRIA, Grenoble, where he leads a group working on distributed systems and applications. Dr. Hagimont has published over 50 papers in areas of distributed systems, and has been active in the organization of conferences in distributed systems.