Dynamic Configuration of Multimedia Applications

Slim Ben Atallah, Oussama Layaida, Noel De Palma, and Daniel Hagimont

INRIA, SARDES Project
ZIRST-655, Avenue de l'Europe - 38334 Montbonnot Saint-Ismier Cedex – France
Slim.Benatallah@inria.fr

Abstract. Streaming multimedia applications, such as video on demand or conferencing are increasingly deployed in heterogeneous and mobile environments including Workstations, PDAs, mobile phones, etc. These applications are very resource demanding and in general, they need to be dynamically adapted when executed on low capability terminals. The proxy-based content adaptation approach is well suited to transparently adapt in real time multimedia data on intermediate nodes without modifying the application. In this paper, we report on experiments on dynamic configuration of such proxies by using a configuration language called APSL (Adaptation Proxy Specification Language). We developed a configurable proxy allowing adaptation of existing videoconferencing applications, and evaluated the performance benefits of the proxy approach using a *DirectShow*/*COM*-based framework.

1 Introduction

Due to the significant growth of network technologies, distributed computing environments are becoming increasingly heterogeneous. This heterogeneity mainly covers two aspects:

- Hardware/Software. Many mobile hand-held devices have appeared such as PDAs or mobile phones. In general, these devices are characterized by low CPU, memory, network or display capabilities. On another hand, the Internet, the operating systems, middleware environments or software libraries which are used by applications may introduce heterogeneity in terms of availability, reliability, latency, network bandwidth, network protocols, data encoding formats, …
- Users' requirements. The potential users of a given application may have very different needs while using the application in a particular context. These requirements may also vary for a same user between different application runs. Moreover, it is important to consider the point of view of many different principals such as a service or content provider, a service client, a network operator, etc.

An adaptation to hardware/software requirements may consist to adapt the encoding format of a streamed video into the unique format supported by an end user terminal. An adaptation to user requirements could consist in selecting in a multi-user videoconferencing session the participants to be displayed on a terminal (e.g. all the participants or one favorite participant). In both cases, the application has to be adapted according to the requirements.

A. Marshall and N. Agoulmine: MMNS 2003, LNCS 2839, pp. 46–63, 2003.

Since it is not realistic to develop, for each application, multiple versions that respond to several constraints, we propose to tackle the various forms of heterogeneity by the development of adaptive distributed multimedia applications. Our objectives are therefore two-fold:

1. Provide distributed multimedia applications with *dynamic adaptation*; that is the ability for applications to adapt their behavior, at launch-time or run-time, to various context constraints.
2. Propose *transparent adaptation* that does not require any modification of the original applications' code.

One important motivation is to be able to adapt applications without any modification to the original code. More importantly, we don't want to reconsider the legacy software (web servers, videoconferencing tool, video players) installed on end-user machines. In order to achieve this goal, we follow a proxy-based architecture where, in a distributed client/server multimedia application, a proxy site between the client and the server is responsible for intercepting interactions (streams) between the two parties and performing the adaptation. An adaptation consists in modifying the content of the stream which traverses the proxy.

The second important motivation is to respond to the various constraints previously enumerated. Since these constraints may be very different and are only known at runtime, we need to dynamically configure the adaptation software on the proxy nodes according to these constraints. Therefore, our approach is to provide a framework for *dynamic configuration of a multimedia proxy*.

We conducted several experiments on a *DirectShow* [5] platform. With its widespread distribution and its efficient implementation, this platform is now considered as a reference platform for building multimedia applications. These experiments aim at validating this approach by implementing several applications and adaptation scenarios. We also evaluated the obtained performance benefits. The lessons learned from this experiment are the following:

- Performing multimedia application adaptations on proxy machines is a means to take into account a very broad range of requirements coming of the hardware, software and user environments, without reconsidering the applications installed on end-user machines.
- Dynamically-configured component-based adaptations provide the required flexibility to deal with the heterogeneity of the requirements.
- The proposed configuration approach implemented on the *DirectShow* environment allows real time adaptation of video contents providing a good tradeoff between flexibility and performance.

This paper is structured as follows. Section 2 discusses the related work. Section 3 presents the configuration of proxies. Our experiments are then detailed in section 4. Section 5 presents results of a performance evaluation. Section 6 concludes the paper while enumerating some perspectives to this work.

2 Related Work

Many component-based environments have been proposed to support the building of architecture-based applications. These environments exploit software architectures to provide dynamic configuration mechanisms in order to adapt applications according to execution constraints [3], [15], [13]. However, very few projects have validated the effectiveness of dynamic configuration with resource intensive applications such as multimedia applications.

Kon [14] describes a framework which allows configuring proxy nodes (called Reflectors) in order to adapt the distribution of multimedia data. However, the main focus is on the adaptation of network routing. Blair [4] describes a framework to adapt multimedia applications for mobile environments. Some proposed scenarios (related to video transcoding) are similar to ours, but their implementation relies on a CORBA platform and they don't report any performance evaluation.

Many projects have addressed the issue of the adaptation of multimedia data delivery according to the application context. A first class of solutions addresses this issue by modifying the servers, the network protocols or the encoding formats used to deliver multimedia data. Sisalem [22] extends servers in order to adapt the emitted streams according to clients' requirements. In [18], McCanne uses multi-layered encoding and transmission, which consists in dividing the video into several cumulative layers, each corresponding to an increment of quality. By selecting the appropriate number of layers, receivers can control the amount of video bandwidth they consume, and solve bandwidth heterogeneity problems. However, these solutions only address network bandwidth; adaptations do not consider clients' hardware limitations and software incompatibilities. Moreover, they require modification of the software installed on Internet hosts (clients and servers).

The alternative solution uses intermediate nodes (proxies) inside the network to make additional treatments on media streams. These entities can be deployed for example by ISPs across the Internet, by network operator or individual users in their private networks. Multimedia content is adapted dynamically so that it matches the variable network or host capacities, without requiring modification on the end machines. Fox [7] presents the advantages of infrastructural proxies and proposes design principles to effectively address heterogeneity problems. Various research works have been made in this area [1], [20], [23]. Most of existing works addressed only adaptations of discrete media, such as HTML pages and images [10][16]. Some projects proposed gateways to adapt video streams. For example, VGW [1] can transcode RTP video streams from high bit-rate MJPEG format into 128 Kbps H.261 video streams which are more suitable for MBone sessions. In [24], an MPEG specific proxy-based content adaptation gives more priority to I-frames than P and B-frames by selecting frames to drop when congestion occurs. A RTP to HTTP gateway, described in [12], interconnects a multicast network with the World Wide Web, and enables Web client to receive video streams from multicast conference sessions. However, all these projects were proposed to solve a particular problem and focused on a specific encoding format or protocol conversion.

Our objective is to provide proxies with the flexibility required to implement any of these adaptations. Moreover, such a flexibility aims at providing content based adaptation of video stream in order to take into account both QoS problems and user

requirements. In this vein, we propose to use a component-based framework which allows dynamic configuration of a context-specific adaptation proxies.

3 Dynamic Configuration of Adaptation Proxies

3.1 Proxy Implementation

Our goal is to dynamically configure adaptation proxies. We provide a proxy configuration language to define the adaptation and also a flexible configuration programming interface allowing deployment and reconfiguration of proxies. We aim at providing a flexible solution of configuration which allow deployment and configuration of proxies on several multimedia environments. We experiment the implementation of dynamic configuration on the Microsoft's *DirectShow 8.0* environment. This environment provides programmers with an efficient toolkit for building multimedia applications. It is based on the COM component model 8 and it provides abstractions for the manipulation of multimedia data, such as filters, pins and filter graphs.
A filter is the basic building block in *DirectShow*. It is a software component offering a specific multimedia-related functionality, such as capturing, encoding/decoding and rendering multimedia data. Using inheritance techniques, programmers can build additional filters. Several filters can be combined (i.e. interconnected) in order to form a filter graph that represents a particular configuration. The interconnection of several filters is possible thanks to pins, which are filters' input and output ports.

3.2 Components

All needed functions for networking and multimedia data processing are provided by separate basic components. A component is characterized by a set of input and output stream interfaces and a processing unit. Moreover, each *DirectShow* component (Filter) provides one or more configuration interfaces which allow configuring the component. These basic components are the following:

- Networking components: They implement standard Internet protocols used for multimedia streaming, such as HTTP1 and RTP.
- Decoder/Encoder components: They aim at compressing/uncompressing the data into a chosen intermediate format (e.g. RGB24, YUY2). We provide different encoder/decoder components which can be used to encode/decode standard multimedia format such as H261, H263, MPEG1, MPEG2 or MJPEG.
- ***Transformer components:*** Transformer components implement the basic content based adaptation code. For instance, a transformation component can receive as input a video stream in YUY2 format, resize it and deliver the modified video as output. Each transformer component provides a very basic adaptation on a stream in an intermediate format. Complex stream transformations can be built by

[1] We had to implement networking components for HTTP as they were missing in DirectShow.

combining several basic components. Below are examples of transformer components that we implemented:

- *Image-scaling components* resize video frames, which is useful to adapt a stream for devices with limited display capacities. They are sometimes required to enable transcoding operations, for example MPEG videos may be transmitted in any size while H.261 videos require predefined sizes such as CIF, QCIF or SQCIF.
- *Color-space-scaling components* reduce the number of entries in the color space, for example from 24 to 12 bits, gray-scale or black-and-white.
- *Data insertion components* can be used to insert an image or a text in a video. We used such components to integrate commercials, subtitles in video and textual notifications.
- *Mixer Component* allows building of a mixed video stream resulting from several input sources. The resulting video stream is an *(NxM)* matrix. Each element of this matrix results from an image-scaling adaptation of a particular stream.
- *Multiplexors/Demultiplexors* are used to aggregate/separate audio and video data in a multimedia stream. For instance, an MPEG Multiplexor allows merging an MP3 audio and an MPEG-1 video in a MPEG2 stream. In the presentations of our scenarios, we sometimes omitted these components since our primary focus is on video adaptations.
- *Duplicator components* are used to replicate an output media stream. Duplication is useful when a stream has different targets with different requirements. Duplicators are used in the videoconferencing application further considered.

Data insertion, mixer, duplicator and some networking components were not provided by the original *DirectShow* framework. These additional components and also the configuration manager code are built on top of this framework to make possible instantiation of proxies performing content-based adaptations.

3.3 Adaptation Sessions

A session is instantiated as a graph of interconnected components that implements the adaptation process. The adaptation process is built from the basic components presented above (receivers, decoders, transformers, encoders …). These components are configured and interconnected together to achieve the transformation of the multimedia stream. The configuration of sessions provides the flexibility required by the adaptation process to fulfill the application needs.
The adaptation process is split up into several steps. The input stream is decoded into an intermediate representation, and then transformed and delivered to the encoder, which produces an adapted stream in output. During this process, adaptations can be applied at different levels in the data path. Fig 1 describes the configuration of a session at a high-level.

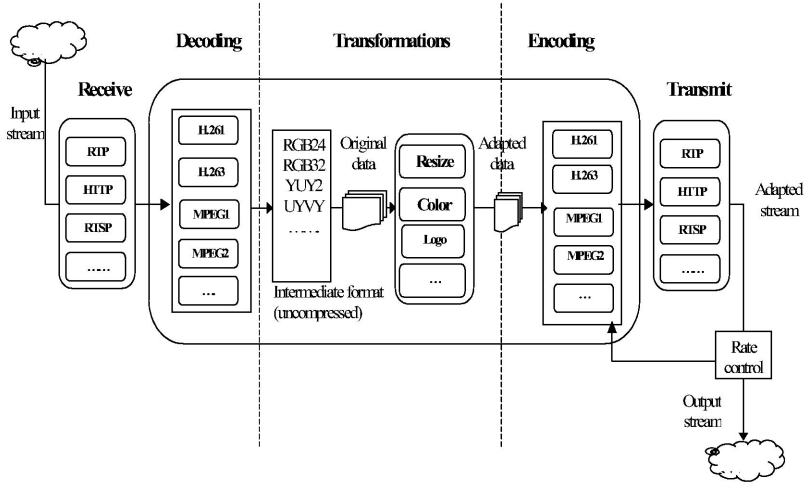

Fig. 1. Video transcoding scheme

Multimedia data is generally transmitted with application-level protocols such as, HTTP, RTP, etc. When configuring a session, an appropriate component[2] is chosen to receive a media stream from the network and to deliver it to the appropriate decoder. At this level, the appropriate decoder component is configured with the intermediate format in which uncompressed data will be represented. This intermediate format allows us to perform additional treatment on data that cannot be performed in a compressed format. Support for multiple intermediate formats allows us to make an optimized configuration to perform these effects (for example, resizing an image in YUY2 format is faster than in RGB format).

At the intermediate level, the data can be transformed in various ways by combining transformer components together in the session's configuration. Changing the interconnections between these transformer components allows customizing the adaptation process according to the requirements.

At the encoder level, an encoder is selected and configured to offer the best-suited data rate that matches network and receiver's states and capacities. The target data rate is obtained by modifying the rate of encoded frames or by degrading the encoding quality. The obtained stream is sent using the protocol used by the network target independently from the protocol used to receive the original data from the server.

3.4 APSL: Adaptation Proxy Specification Language

In order to help the definition of a session, we propose an XML-based specification language called APSL, which allows describing several QoS parameters and User requirements such as input video format, network capabilities, terminal capabilities,

[2] In fact, one or several networking components may be necessary to receive video streams. As described further in the paper, a videoconference adaptation session may rely on several networking components to receive several input streams.

connection protocols, etc. An APSL specification may be composed of the following definitions:

- **INPUT** : defines a list of members. Each member describes a particular input source of the proxy. Attributes of an input member are : PROTOCOL, USER, TERMINAL, and DATA.
- **OUTPUT**: defines a list of output destinations members. Each member describes a particular adapted target streams. OUTPUT members are also defined using the same INPUT member attributes.
- **PROCESS**: defines the proxy architecture. We use a directed graph model to define the adaptation process on proxies. Each graph node represents a particular component. Components are bound using input/output PIN connections. The Document Type Definition of APSL is detailed in Fig 2.

```
<!ELEMENT APSL (INPUT, OUTPUT, PROCESS)>  ← APSL element list
definition
<!ELEMENT INPUT (MEMBER+)>← INPUT member list definition
<!ELEMENT OUTPUT (MEMBER+)>      ←OUTPUT member list definition
<!ELEMENT MEMBER (PROTOCOL, USER, TERMINAL, DATA)> ←
INPUT/OUTPUT member definition
<!ATTLIST MEMBER ID ID #REQUIRED>
<!ELEMENT PROTOCOL (NAME, DESCRIPTION, ARGUMENT*)>
<!ELEMENT USER (LOGIN, PROPERTIES)>
<!ELEMENT LOGIN (#PCDATA)>
<!ELEMENT PROPERTIES (#PCDATA)>
<!ELEMENT ARGUMENT (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT TERMINAL (CPU, DISPLAY, NETWORK)>
<!ELEMENT DISPLAY (XSIZE, YSIZE, COLORDEPTH)>
<!ELEMENT CPU (#PCDATA)>
<!ELEMENT NETWORK (#PCDATA)>
<!ELEMENT XSIZE (#PCDATA)>
<!ELEMENT YSIZE (#PCDATA)>
<!ELEMENT COLORDEPTH (#PCDATA)>
<!ELEMENT DATA (TYPE, CODEC+)>
<!ELEMENT TYPE (#PCDATA)>
<!ELEMENT CODEC (#PCDATA)>
<!ELEMENT PROCESS (COMPONENT*)>  ← PROCESS element definition
<!ELEMENT COMPONENT (PIN+)>
<!ATTLIST COMPONENT
  ACTION (RESIZE | DUPLICATOR | H.261Encoder | MPEGEncoder |
H.263Encoder | MIXER) #REQUIRED ID ID #REQUIRED>
<!ELEMENT PIN (#PCDATA)>
<!ATTLIST PIN ID ID #REQUIRED
LINK IDREF #IMPLIED
DIRECTION (OUTPUT | INPUT) #REQUIRED>
```

Fig. 2. APSL DTD.

3.5 Proxy Configuration Architecture

A proxy can be configured through an APSL specification which describes the initial configuration, including the proxy's process architecture and the members' attributes. This APSL specification is interpreted by an **APSL Engine** which is responsible for the instantiation of the associated adaptation session.
At a lower level, a **Configuration Manager** provides all the functions required to instantiate and manage adaptation sessions. The API of the Configuration Manager is invoked by the APSL Engine in order to instantiate a session based on an APSL specification. The Configuration Manager implementation directly relies on DirectShow/COM.
However, an adaptation session may have to be dynamically adapted in order to respond to variations of the execution constraints (e.g. available resources). For this reason, the API of the Configuration Manager is exported to allow direct management of the adaptation session. It allows visiting and adapting the component-based architecture of a session, or modifying the attributes of the session's components.
The overall structure of the proxy environment is shown in Fig 3.

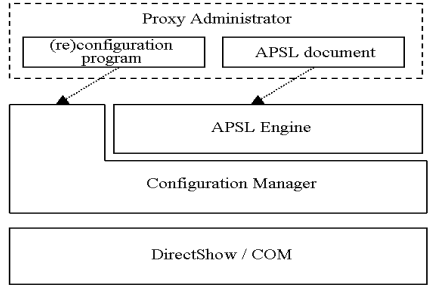

Fig. 3. Overall structure of the proxy environment.

The administrator of the proxy can configure an adaptation session by providing an APSL document which describes the required adaptation. He can also directly use the Configuration Manager API to configure the required adaptation session. And finally, this Configuration Manager API can be used to reconfigure an adaptation session, i.e. to modify it in response to variations in the execution conditions.

4 Experiments

Our objective is first to validate the approach described above with the implementation of realistic scenarios and second to show that it can be efficiently instantiated, thus combining flexibility and performance.
We modeled an experimental environment involving several multimedia applications and mobile handheld devices. The platform is based on PC workstations (PIII 700Mhz-256MB for VoD scenario, PIV 1800 Mhz-512MB for Videoconferencing scenario) interconnected with an Ethernet Local Area Network (Ethernet 100 Mbps)

and mobile PDA (IPaq, ARM Processor 200 MHz, 32 MB RAM, Windows CE 3.0) devices connected through a 802.11 Wireless access points. We experimented with two applications: a video on demand service and a videoconferencing system. The VoD application relies on a web server which hosts several MPEG movies encoded with high quality parameters. The videoconferencing application distributes real-time video streams using VIC [17], an application from University of California, Berkley. The client side includes standard applications offering basic multimedia players; on the PDAs, we used PocketTV [19] for streaming MPEG-1 movies using HTTP and VVP [25] for real-time H.261 streaming.

4.1 VoD Scenario: Adaptation for Hardware/Software Capabilities of the PDA

In the first scenario, we consider the video on demand application for mobile handheld devices (PDAs). Due to their limited processing, display and network capacities, PDAs are only able to efficiently render streams with specific properties (frame size, colors, quality factor and encoding format). To deal with such hardware limitations, we configure a dedicated adaptation proxy. When the client sends an HTTP request to the proxy (using the proxy URL instead of the original URL), the VoD adaptation proxy parses it in order to extract the client properties and it invokes the configuration manager API to instantiate a session according to the received properties (the HTTP request fields give the following client's properties: Accept-encoding, color depth and frame size). Fig 4 gives the composition of the session which adapts an original MPEG stream into an MPEG stream with smaller resolution and color depth in order to fit PDA's display capacities.

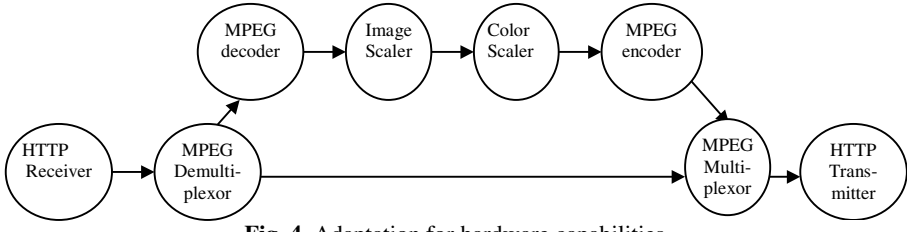

Fig. 4. Adaptation for hardware capabilities

The instantiated session includes two networking components for receiving and transmitting the HTTP streams. Between them, six additional components are inserted. First, an MPEG demultiplexor separates the MPEG audio and video into two streams, an MP3 audio stream and an MPEG-1 video stream. The video stream is handled by an MPEG-1 decoder component, which uncompresses data into YUV video frames. Then, an image-scaler component resizes video frames to QCIF (176*144), transformed by a color down-scaler component into 16 gray-scale colors. Notice that the quality factor of the encoder can also be dynamically adjusted. A similar scenario is used to illustrate adaptation for software capabilities on the PDA. In this context, the proxy transcodes the original stream into H.261 and forwards it to the client.

In addition to the transcoding operation, a third party may want to integrate other services such as the insertion of commercial advertisement or personalized subtitles in the video content.

4.2 Adapting VIC/VVP Videoconferencing Application: Video Stream Mixing and Bandwidth Adaptation

Videoconferencing applications often involve more than two participants, each with its own encoding format and terminal capabilities. The scenario that we implemented focuses on the following problems:

- *A client machine requires a high bandwidth to receive multiple streams and a high processing capacity to decode and synchronize them before display.*
- *On the other hand, VVP which is the VIC version for PDAs does not provide support for multiple streams visualization. We modified the conference architecture (without any modification of VVP/VIC) in order to introduce a proxy conferencing server which receives a stream from each participant, mixes the streams in a single video stream which is sent to all participants.*

VIC and VVP can be used in a multicast mode or as peer-to-peer applications. When used with multicast mode, VVP receives all incoming video streams but can only display one at the same time (due to the limited display capacity of the PDA). The objective of our adaptation is first to reduce the used bandwidth (by emitting a single stream from the proxy to the client machine), and second to allow display of all participants' videos on the VVP user interface.

The scenario that we consider describes a multiparty videoconference between five terminals: A and B are two workstations running VIC with H.263, C and D are also two workstation running VIC with H.261 CIF resolution, and E is a PDA running VVP to just receive video in H.261 and QCIF resolution. The conference starts between A, B, C and D. The participant connected via PDA "E" joins the conference at a later time. For each Workstation participant, the proxy opens one incoming and one outgoing stream. The incoming stream contains the media of that user and the outgoing stream is the result of mixing all media streams into one stream, requiring less bandwidth and less computing on the client. Fig 5 describes the configuration graph resulting from parsing APSL configuration of the proxy.

For each incoming stream, the proxy instantiates an RTP source filter and the appropriate decoder component. Streams received from A and B are resized into QCIF. A central mixer component receives the three videos streams and produces a mixed video stream in CIF size (4 QCIF quarters, one empty). As participants use two different encoding formats, the mixed video is duplicated into two streams with a duplicator component. The first one is encoded in H.263, duplicated again into two streams and sent to participants A and B. The second one is resized to QCIF, encoded in H.261 and sent to participant C and D. When user E joins the conference, the proxy requests a new receiving branch (dotted bag on the left side of the figure). As this participant requires H.261 encoding, the proxy inserts a duplicator after the H.261 encoder and a new RTP transmitter is created to send the mixed stream to E (dotted bag on the right side of Fig 5).

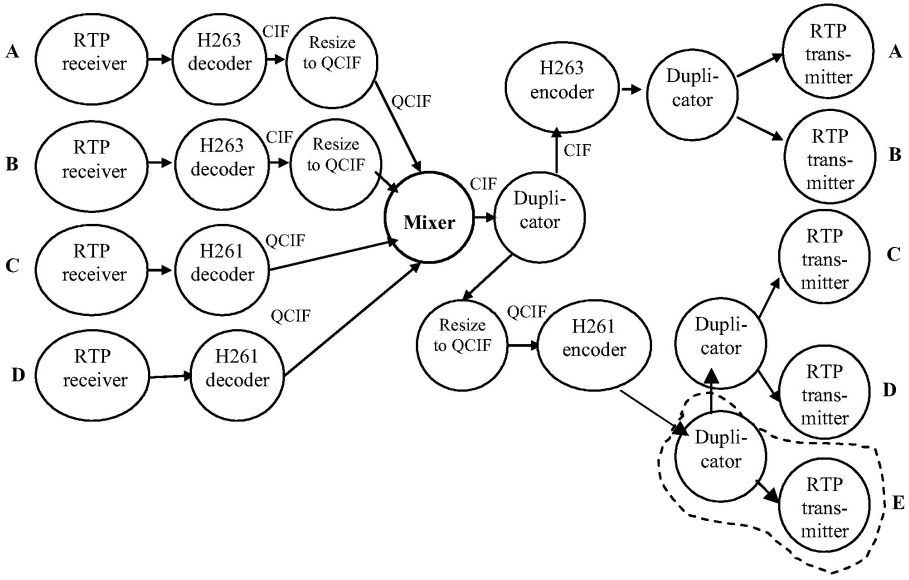

Fig. 5. Dynamic configuration of a videoconferencing proxy

The use of the duplicator components optimizes the configuration in order to prevent redundant tasks. Rather than serving each user independently (e.g. with per-user encoders and transformers), the configuration manager looks for an existing output stream providing the same properties as those requested by the arriving participant.
Fig 6 shows a screen shot of the adapted VVP videoconferencing application in which 4 video streams are mixed and rescaled in order to be displayed on the PDA.

Fig. 6. VVP adapted user interface

4.3 APSL Specification for Videoconferencing Adaptation Scenario

In order to configure the videoconferencing proxy, we used an appropriate APSL specification (cf. Fig 7) which defines for each end-user terminal, the video format, the network connections, and screen size.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE APSL SYSTEM "C:\inria\Sardes\MAD\code\MM\APS\apsl.dtd">
<APSL>
    <INPUT>
        <MEMBER ID="M1">
            <PROTOCOL>
                <NAME>RTP</NAME>
                <DESCRIPTION>VICclient</DESCRIPTION>
            </PROTOCOL>
            <USER>
                <LOGIN>USER1</LOGIN>
                <PROPERTIES>194.199.25.10</PROPERTIES>
            </USER>
            <TERMINAL>
                <CPU>PDA(StrongARM)</CPU>
                <DISPLAY>
                    <XSIZE>176</XSIZE>
                    <YSIZE>144</YSIZE>
                    <COLORDEPTH>32bits</COLORDEPTH>
                </DISPLAY>
                <NETWORK/>
            </TERMINAL>
            <DATA>
                <TYPE>Video </TYPE>
                <CODEC>H.261</CODEC>
            </DATA>
            </MEMBER>
        <MEMBER ID="M2">
        </MEMBER>
        <MEMBER ID="M3">
        </MEMBER>
        <MEMBER ID="M4">
        </MEMBER>
    </INPUT>
    <OUTPUT>
        <MEMBER ID="M5">
                ...
        </MEMBER>
        <MEMBER ID="M6">
        </MEMBER>
        <MEMBER ID="M7">
        </MEMBER>
        <MEMBER ID="M8">
        </MEMBER>
    </OUTPUT>
    <PROCESS>
        <COMPONENT ID="C1" ACTION="RESIZE">
                <PIN ID="P2" DIRECTION="OUTPUT" LINK="C3" />
                <PIN ID="P1" DIRECTION="INPUT" LINK="M1" />
        </COMPONENT>
        <COMPONENT ID="C2" ACTION="RESIZE">
                <PIN ID="P3" DIRECTION="INPUT" LINK="M2" />
                <PIN ID="P4" DIRECTION="OUTPUT" LINK="C3" />
        </COMPONENT>
        <COMPONENT ID="C3" ACTION="MIX">
                <PIN ID="P5" DIRECTION="INPUT" LINK="C1" />
                <PIN ID="P6" DIRECTION="INPUT" LINK="C2" />
```

```
                <PIN ID="P7" DIRECTION="INPUT" LINK="M3" />
                <PIN ID="P8" DIRECTION="INPUT" LINK="M4" />
                <PIN ID="P9" DIRECTION="OUTPUT" LINK="C4" />
            </COMPONENT>
            <COMPONENT ID="C4" ACTION="DUPLICATOR">
                <PIN ID="P10" DIRECTION="INPUT" LINK="C3" />
                <PIN ID="P11" DIRECTION="OUTPUT" LINK="C5" />
                <PIN ID="P12" DIRECTION="OUTPUT" LINK="C6" />
            </COMPONENT>
            <COMPONENT ID="C5" ACTION="H.263Encoder">
                <PIN ID="P13" DIRECTION="INPUT" LINK="C4" />
                <PIN ID="P14" DIRECTION="OUTPUT" LINK="C8" />
            </COMPONENT>
            <COMPONENT ID="C6" ACTION="RESIZE">
                <PIN ID="P15" DIRECTION="INPUT" LINK="C4" />
                <PIN ID="P16" DIRECTION="OUTPUT" LINK="C7" />
            </COMPONENT>
            <COMPONENT ID="C7" ACTION="H.261Encoder">
                <PIN ID="P17" DIRECTION="INPUT" LINK="C6" />
                <PIN ID="P18" DIRECTION="OUTPUT" LINK="C9" />
            </COMPONENT>
            <COMPONENT ID="C8" ACTION="DUPLICATOR">
                <PIN ID="P22" DIRECTION="INPUT" LINK="C7" />
                <PIN ID="P23" DIRECTION="OUTPUT" LINK="M6" />
                <PIN ID="P24" DIRECTION="OUTPUT" LINK="M5" />
            </COMPONENT>
            <COMPONENT ID="C9" ACTION="DUPLICATOR">
                <PIN ID="P22" DIRECTION="INPUT" LINK="C7" />
                <PIN ID="P23" DIRECTION="OUTPUT" LINK="M6" />
                <PIN ID="P24" DIRECTION="OUTPUT" LINK="M5" />
            </COMPONENT>
        </PROCESS>
</APS>
```

Fig. 7. APSL specification for VIC/VVP videoconferencing adaptation.

5 Performance Evaluation

We evaluated the benefits of dynamic video content adaptations for the performance of the PDA. Evaluation is performed for two application settings:

Table 1. Performance of the PDA with the VoD application

Format	Size	Maximum frame rate (fps)
MPEG1	640*480	3.8
MPEG1	320*240	9.2
MPEG1	176*144	21.4
H.261	176*144	24.5

The video on demand application. A video file is available on a Web server and accessed from the PDA. The video format is MPEG-1 and the video size is 640*480

(we write it MPEG-1/640*480). The evaluated proxy adaptations respectively convert the initial stream into MPEG-1/320*240, MPEG-1/176*144 and H.261/176*144. The video is always encoded at the same frame rate and data rate (25 frames/s, 250 Kbits/s).

The videoconferencing application. It distributes a video stream in H.261-CIF (352*288) at 25 frames/s and the best quality factor. The evaluated proxy adaptation converts 4 initial streams into one mixed H.261-QCIF (176*144) stream with a lower quality factor (85 %).

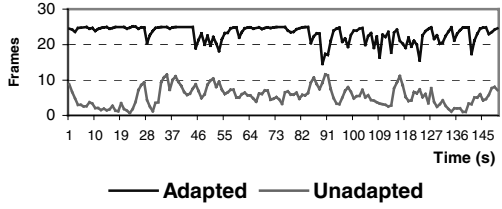

Fig. 8. Performance of the PDA with VVP application

On Table1 and Fig 8, we observe that the PDA can hardly display a large size video because it has to decode and resize the video frames to display them. As a PDA does not have sufficient capacities to perform these operations in real time, frames are displayed at a very low rate. However, when the size decreases, the frame rate increases accordingly, in particular for the QCIF size (176x144), which is probably the best-suited size for handheld devices such as PDAs.

In Table1, we also observe that the frame rate increases when we change the encoding format from MPEG to H.261, which decoding is less demanding.

Power Consumption of the PDA

Power consumption on the PDA, i.e., energy consumption, is an important problem for mobile devices with intrinsic power limitations. Indeed, unlike desktop PCs with permanent power supply, mobile devices such as PDAs have limited power autonomy. Several research works have studied the relationship between power consumption and multimedia applications [7, 21]. Analysis and measurements have demonstrated existing dependencies with the size, color depth and encoding format of data. In order to measure the power consumption on the PDA, we used the notification given by the PDA's battery driver when the battery level changes by ±10%. We therefore run the multimedia application for a long period (up to one hour) and measured the average period at which the battery level changes. We performed these measurements with the VoD application and the proxy adaptations described in the previous section. The results are given in Table 2.

We observe that power consumption also depends on the video size and its encoding format. When the proxy is configured to reduce the video size and to use a cheaper encoding format, displaying the video requires less treatments and consequently it consumes less energy.

Table 2. Power consumption of the PDA with different formats and sizes

Format	Size	10% Power consumption
MPEG1	640*480	every 12 minutes
MPEG1	320*240	15 minutes
MPEG1	176*144	19 minutes
H.261	176*144	24 minutes

Network Bandwidth

Proxy adaptations can also be beneficial in order to save network bandwidth. In order to evaluate this, we used the videoconferencing application and we measured on the PDA the data bit-rate and the packet loss. Data bit-rate gives the amount of data transferred on the network. Packet loss can be caused by network congestion or by PDA's processor overload. Lost packets due to processor overload are delivered to the PDA but discarded as they could not be processed in time. These discarded packets also result in wasted network resources as they are discarded after reaching the PDA.

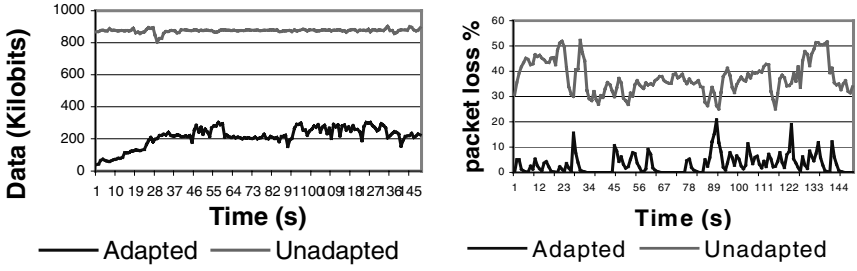

Fig. 9. Impact of adaptation on network traffic

Results are given in Fig 9; in both measurements (data rate and packet loss), the upper curve corresponds to a transmission without proxy adaptation (i.e. using a multicast data communication scheme). On the left side of the figure, we observe that the adapted stream consumes less bandwidth because resizing the video reduces considerably the amount of data transmitted on the network.

On the right side of the figure, we observe that without adaptation, packets are lost at a high rate varying between 30 and 50% (due to PDA's processor overload). This results in a displayed frame rate lower than 11 frames/s (Fig 9) and thus a poor quality presentation. With proxy adaptation, packet loss is kept under 10% and frames are displayed at a rate close to the original.

Performance of the Proxy

The ability of a proxy machine to support one or several sessions which adapt multimedia streams in real time is a critical issue. To evaluate this, we measured the processor load on the proxy in function of the number of sessions (Fig 10). Each session is composed of networking components, an MPEG decoder, a resize component (from 320*240 to CIF) and an H.261 encoder.

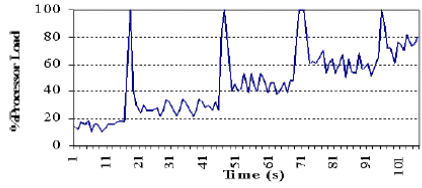

Fig. 10. CPU load on the proxy

Results show that a session consumes on the average 13 % (on a PIII 700 Mhz with 256 MB) of the processor CPU resource. Notice here that these results depend on the number of used codecs, the decoding/encoding formats and the treatments applied to the video in the session. We consider here a very common adaptation of the VoD application, i.e. when we adapt a video stream for a PDA (MPEG to H.261 conversion, and resize to 176*144). We also observe a peak when the proxy creates a new session. This peak corresponds to the creation and the configuration of the session. These results seem acceptable as we used a medium range PC to run the proxy software. In addition, for a large number of clients, the load of the proxy could be balanced between several machines as proposed in [4].

6 Conclusion and Perspectives

In this paper, we reported on an experiment which consisted in evaluating the benefits of dynamic content-based adaptations for distributed multimedia applications. Adaptations are used to face the increasing heterogeneity of today's distributed computing environment: heterogeneity of the hardware, the software and the user preferences. Adaptations are transparently performed on network intermediary nodes called proxies. Finally, adaptations are dynamically configured according to runtime constraints, thanks to a component-based middleware. We used *DirectShow* as an implementation base layer for conducting this evaluation. lessons learned can be summarized as follows:

- Proxy-based approach allow an implementation of adaptations without reconsidering the applications installed on end-user machines. We have implemented application scenarios that demonstrate this possibility without any modification on reused software such as Web servers, videoconferencing tools, or video viewers.
- Dynamically-configured content-based adaptations using APSL provide the required flexibility to deal with the diversity of environment parameters. Our application scenarios (Vod and videoconference) experimented in different

execution environments (PCs on a LAN, PDAs on a WLAN) demonstrate this flexibility.
- *DirectShow* can be extended to provide a well-suited support for dynamic configuration and reconfiguration of adaptation proxies. Its performance allows real time adaptation of video contents. We therefore provide a good tradeoff between flexibility and performance.

We are currently working on the extension of our framework for supporting session establishment protocols such as SIP [21] or H.323 [12] sessions.
At the moment, proxy configurations have to be programmed using the *DirectShow* API. We implemented a configuration manager which dynamically configures the proxy according to a unique APSL configuration file. However, It would be interesting to deploy the adaptation process on several sites and use several APSL files. We made few experiments with dynamic reconfiguration (during execution) of the proxy according to available network and terminal resources. However, we lack a deeper evaluation of dynamic reconfiguration of several proxies cooperating to take into account adaptation. Such experiments would require to integrate a monitoring service in our framework.

References

1. E. Amir, S. McCanne, Z. Hui. An Application Level Video Gateway. Proc. of ACM Multimedia'95, San Francisco, Nov 1995 .
2. Baldonado, M., Chang, C.-C.K., Gravano, L., Paepcke, A.: The Stanford Digital Library Metadata Architecture. Int. J. Digit. Libr. 1 (1997) 108–121.
3. L. Bellissard, S. Ben Atallah, F. Boyer, and M. Riveill. Component-Based Programming and Application Management with Olan. In Proceedings of Workshop on Distributed Computing Systems, pages 579–595, May 1996.
4. G. Blair, A. Andersen, L. Blair, G. Coulson, and D. S. Gancedo, "*Supporting dynamic QoS management functions in a reflective middleware platform*," IEE Proceedings – Software, 2000.
5. Microsoft DirectX (version 8.0): Microsoft DirectShow, Online Documentation: http://msdn.microsoft.com/directx/.
6. A. Fox, S.D. Gribble, Y. Chawathe, E.A. Brewer, and P. Gauthier. Cluster Based Scalable Network Services. In Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, October 1997.
7. 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.
8. A. Gordon, A. I. Gordon. *The COM and COM+ Programming Primer*. Prentice Hall, March 2000.
9. C.J Hughes, J. Srinivasan, and S.V. Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. Proceedings of the 34th International Symposium on Microarchitecture, Dec 2001.
10. IBM Inc. Internet Transcoding for Universal Access, Sep. 2000. http://www.research.ibm.com/networked_data_systems/transcoding/index.html.
11. ITU-T Recommendation H.261: Video codec for audiovisual services at p x 64 kbit/s. Geneva, 1990, revised at Helsinki, Mar. 1993.
12. M. Johanson, An RTP to HTTP video gateway. In Proceedings of the Tenth International World Wide Web Conference, Hong Kong , May 2001.

13. F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhães, R. H. Campbell, Monitoring, Security, and Dynamic Configuration with the DynamicTAO Reflective ORB, Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware'2000.
14. F. Kon, R. H. Campbell, K. Nahrstedt, Using Dynamic Configuration to Manage A Scalable Multimedia Distribution System, Computer Communications, 2000.
15. J. Magee, J. Kramer, M. Sloman. Constructing Distributed Systems in Conic. IEEE Transactions on Software Engineering, 15(6):663–675, June 1989
16. A. Maheshwari, A. Sharma, K Ramamritham et P. Shenoy. TranSquid: Transcoding and Caching Proxy for Heterogenous E-Commerce Environments, Proceedings of the 12th IEEE Workshop on Research Issues in Data Engineering (RIDE '02), San Jose, California, Feb 2002.
17. S. McCanne and V. Jacobson. VIC: A exible framework for packet video. Proc. of ACM Multimedia'95, Nov 1995.
18. S. McCanne, V. Jacobson, M. Vetterli. Receiver-Driven Layered Multicast. In SigComm'96, Stanford, CA, Aug 1996.
19. MPEG Movie Player for PocketPC, (http://www.pockettv.com), 2000.
20. R. Rejaie, H. Yu, Mark Handley, D. Estrin. Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet. Proceedings of the Conference on Computer Communications (IEEE InfoCom), Mar 2000.
21. 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.
22. D. Sisalem, H. Schulzrinne. The loss-delay based adjustment algorithm: A TCPfriendly adaption scheme. Proc. NOSSDAV'98, Jul 1998.
23. J. Smith, R. Mohan, C. Li. Transcoding Internet Content for Heterogeneous Client Devices. IEEE Conference on Circuits and Systems, Monterey, Jun 1998.
24. M. Hemy and al. MPEG system Streams in Best-Effort Networks. Packte Video'99, Cagliari, Italy, April 1999.
25. M.Thorson. VIC viewer for PocketPC, Available Online: http://www.oncoursetech.com/video/default.htm), Apr 2001.
26. W. Yuan, K. Nahrstedt, X. Gu. Coordinating Energy-Aware Adaptation of Multimedia Applications and Hardware Resource. Proceedings of the 9th ACM Multimedia Multimedia Middleware Workshop), Oct 2001.