Adaptive Video Streaming for Embedded Devices[1]

O. Layaida and D. Hagimont
SARDES Project- INRIA Rhône Alpes
655 Avenue de l'Europe, 38334, Montbonnot
FRANCE

Abstract

Recent advances in mobile equipments and wireless communications are making the Internet increasingly heterogeneous in terms of network, hardware and software capacities. Moreover, resources availability on the Internet varies unexpectedly. Thus, providing an efficient access to multimedia services requires that multimedia streams be adapted according to the environment constraints. One approach to this issue is based on the use of intermediate nodes within the network to perform such adaptations (media transformations, protocol conversion, and data transcoding) thus minimizing intrusivity with respect to legacy applications. Many research works have followed this approach, but they lack the flexibility required to fully address the issue.

In this purpose, we designed and implemented a framework for network-based adaptations of multimedia streams. To reach the required flexibility, this framework provides support for dynamic configuration and reconfiguration of adaptations processes. A language called APSL (Adaptation Proxy Specification Language) allows the specification of such (re)configurations. A specification is translated into a component-based software architecture which evolves dynamically according to execution conditions. We describe the implementation of this framework on top of the Microsoft DirectShow® environment and report on a performance evaluation which demonstrates the effectiveness of the approach.

1. Introduction

Recent advances in the areas of mobile equipments and wireless networking have led to the emergence of a wide range of peripherals such as, Personal Digital Assistant (PDA), hand-held computers, Smart Phones, eBooks, etc. Such peripherals are connected to the Internet and 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. In particular, multimedia services generally used on desktop computers such as audio/videoconferencing, IP-telephony and video on demand, may be integrated on these equipments.

However, this is complicated by the heterogeneity of the Internet infrastructure. This heterogeneity appears in network, hardware and software capacities of terminals making the communication between distributed entities difficult if not impossible. Moreover, the execution environments are characterized by unexpected variations of resource availability, such as network bandwidth, CPU load or battery life time. As a result, it becomes difficult to ensure a correct execution of applications during their lifetime.

To solve these problems, it becomes necessary to provide adaptation techniques that help multimedia applications to take into account the characteristics of their environment and its evolution. This field of research has been actively explored during last years; [22] gives a survey and a comparison of existing approaches. One of them consists in integrating media adaptation processes inside the network in order to adapt the multimedia content according to network, hardware or software capacities [7]. A site on which such an adaptation process is deployed is generally called Proxy. An adaptation process may consist in filtering multimedia streams [10], changing their transmission protocol [11], transcoding content [1] or applying spatial and temporal transformations

[1] This work has been funded by a Microsoft Research Innovation Excellence award for Embedded Systems.

[13][21]. The main benefit of network-based adaptation is its non-intrusivity with respect to legacy applications, while benefiting from the processing power available in the data-path. Adaptation tasks can be performed by the proxy transparently to servers and clients, and can be used to lighten the computing load on the client (especially when it is a mobile device).

This class of applications presents a very interesting case of study that expresses two key requirements:

Dynamic configuration: As multimedia applications use various network protocols, encoding format and data properties (resolution, number of colors, quality factor, frame rate, etc.), no assumptions can be made on client characteristics. Consequently, a first requirement concerns the ability to customize adaptation processes according to each application context. This need of *dynamic configuration* is important to deal with a variety of adaptations requirements between arbitrary client devices and multimedia services.

Reconfiguration: In order to take into account changes in the execution environment, an adaptation process must change its behaviour by performing reconfiguration tasks. The way whereby an adaptation process should be reconfigured may vary with the application nature or terminal capacities, making difficult to predict all possible situations. For instance, a real-time videoconferencing is more sensitive to network delay and transmission errors compared to a VoD application streaming pre-recorded video content, and would requires a different congestion control algorithm. In this case, it becomes necessary to provide a *general mean* to achieve reconfigurations.

Our goal is to address these requirements. This paper describes a generic framework enabling the dynamic configuration and the reconfiguration (at runtime) of network-based adaptations. The remainder of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes the framework and its implementation is presented in section 4. Section 5 reports on a performance evaluation of the implemented prototype, and we conclude in section 6.

2. Related work

In this section, we review previous works on providing support for adaptation of distributed multimedia applications. These works are not always focusing on network-based adaptations, but we review them considering their ability to configure and reconfigure application (or proxies). These works can be roughly divided as follows:

Static configuration – no reconfiguration. The first class employs a static configuration addressing only a particular class of applications. Reconfiguration is not taken into account. This approach has been followed in many application-level proxy architectures such as [10][21][11][17][13]. An intermediate proxy is used to perform additional treatments on multimedia streams such as data transformations, protocol conversions, data transcoding, etc. Although these proposals have considered some heterogeneity problems, they focus on specific problems and employ monolithic design of applications. Such solutions cannot face all possible situations coming from the existing heterogeneity.

Static configuration with reconfiguration. In the second class, multimedia applications are configured statically and employ static reconfiguration policies. For example, the MBONE tools VIC [14] and RAT [9] use the RTCP [18] feedback mechanism and a loss-based congestion control algorithm in order to dynamically adapt the transmission rate of media streams to the available bandwidth. This is performed by reconfiguring the encoder properties appropriately (quality factor, frame rate, etc.). In [16], a self-adaptive proxy is proposed to manage QoS for multimedia applications. The limitation of these proposals is that reconfiguration policies are embedded in the application code and cannot be modified without additional development efforts.

Dynamic configuration – no reconfiguration. Some research works have addressed the dynamic configuration issue. In general, an application is built from a specification which defines its structure and its behavior. This approach has been followed in many network-based adaptation architectures, where the main concern is to address different requirements. For example, the Berkeley Active Service Framework [1] uses an agent-based architecture that allows a client application to explicitly start an adaptation process on a given gateway. With a similar goal, Infopipes [2] defines an application-level framework that allows configuring adaptation processes by assembling separately defined components. These proposals bring more flexibility to the configuration of applications. However, they do not consider reconfiguration issues.

Dynamic configuration with reconfiguration. The last class provides dynamic configuration and reconfiguration. CANS (Composable Adaptive Network Services) [8] for example, follows this approach in a network-based adaptation architecture. Reconfiguration tasks are performed by a set of managers responsible for monitoring resources and applying reconfigurations when needed. However, configuration and reconfiguration policies have to be developed using a usual programming language (typically Java or C). A similar approach was followed in TOAST [6]. In both cases, they lack a high-level language support for describing configurations and reconfigurations.

Therefore, we are lacking a **high-level support for dynamic configuration and reconfiguration.** As it has been widely recognized, component-based approach plays an important role in building adaptive applications [3]. Besides the fact that various applications can be composed of several basic components, reconfiguration operations are facilitated through high-level component-related operations like adjusting component's properties, removing/inserting components or modifying their assembly. Complex operations can be made up with a combination of those basic operations, performed in an appropriate order. Accordingly, work around multimedia applications has led to the development of component-based frameworks in such as DirectShow (Microsoft) [15], JMF (Sun) [20] and PREMO (ISO) [5]. Such component-based frameworks are used in CANS [8] and TOAST [6], but these proposals lack flexibility and are in general reliant to a specific class of application.

Our approach is to rely on a component-based infrastructure (presently DirectShow) to enable dynamic configuration and reconfiguration of network adaptation processes. In order to help the definition of configurations and reconfigurations, we provide a high-level language for the description of the software architecture of an adaptation process and the description of the *autonomic* modifications which are applicable to this software architecture.

3. Framework Architecture

Our framework relies on the Microsoft DirectShow®, part of the DirectX architecture which provides advanced support for multimedia applications on various Windows platforms. In this section, we first give a brief description of DirectShow features. We then overview the overall architecture of our framework; detailed description and examples can be found in [12].

3.1. Using the Microsoft DirectShow Architecture

DirectShow follows a data-flow processing paradigm; an application performs any task by connecting together chains of basic components called *Filters* in the same *FilterGraph Manager* (Figure 1).

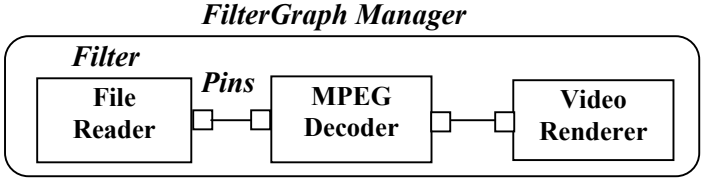

Figure 1. Example of DirectShow graph.

There are three kinds of filters: *source*, *transform*, and *renderer*. Source filters retrieve data from a media source (such as a camera, a local data file, or the internet). Transform filters manipulate data and pass it to the next filter in the graph. Finally, renderer filters output the data to data consumers - such as a video monitor, a data file, or the internet. The arrangement of filters in the filter graph defines the data processing algorithm. Filters are connected through their *Pins* so the *Output Pins* from one filter deliver data to the *Input Pins* of the next components. A connection between two components is only possible when involved components agree the same *media type*, expressed as the MIME type (i.e. audio or video) representation format (e.g. RGB, YUV, etc.) and some data-specific properties (such as resolution).

What makes DirectShow more powerful than other multimedia platforms is that it provides a set of default filters that implement most known multimedia standards encoding formats as well as a SDK easing the addition of custom filters. Taking advantages of this extensibility, we have designed our reconfiguration model upon DirectShow.

3.2. Framework Overview

The framework architecture is depicted on Figure 2. It is structured in three levels: specification, translation and execution. The specification-level defines a language (called APSL) and associated tools for a high-level representation of applications. This language is located at the top of this framework; a specification can be either written by 'human' developers or automatically generated by applications requesting specific multimedia services. In this later case, it may be the result of media session negotiation of usual session-level protocols (e.g. H.323, SIP, RTSP, etc.). The middle-level includes required functions used to translate specifications into entities at the execution-level. It encompasses basic components, based on which various multimedia services can be composed (dotted items in the execution-level illustrate the association with the specification-level).

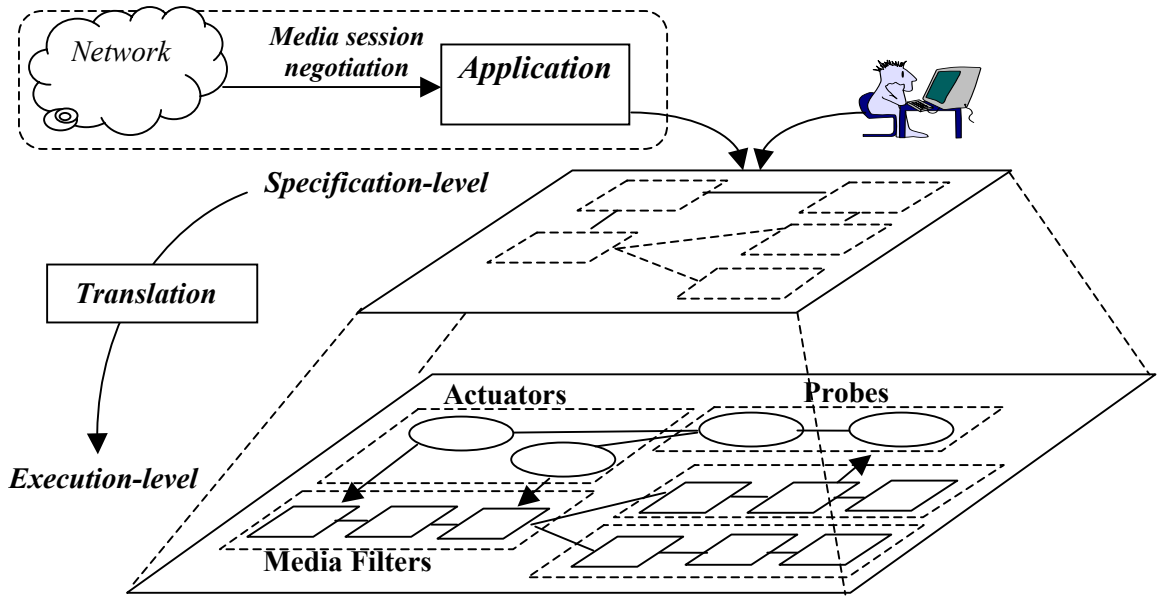

Figure 2. Overall Architecture.

3.2.1. Application Specification

The main concern of the specification language is to find a general model for the description of an application architecture and its reconfiguration policies. APSL (Adaptation Proxy Specification Language) is an XML-based [4] language allowing the specification of adaptation processes. As shown in Figure 3, an APSL specification is composed of three main XML elements described as follows:

- **<Network>** describes incoming and ongoing streams which are respectively defined by **<Input>** and **<Output>** elements.
- **<Process>** describes the adaptation process as an acyclic graph linking several APSL elements (as XML elements, APSL elements are identified using XPath expressions [23]). In fact, each node of this graph defines an elementary multimedia processing unit such as decoding, encoding, mixing, etc. Each node introduces a set of specific XML attributes that determine its properties and common *ilink* and *olink* XML attributes which define the graph edges. At control level, the **<Process>** element is used to instantiate the adaptation components, their properties and also the bindings between these components.
- **<Reconfiguration>** defines the reconfiguration policies that should be applied at runtime. Each policy is defined by a **<Rule>** element which is defined itself by a **<Probe>** element and a set of **<Condition>** elements. Each **<Probe>** element is associated with an APSL element of the graph (as specified in the example in Figure 3, the **<Probe>** element in rule *R1* is associated with the *OUT* element). A **<Probe>** element is composed of one or more **<Event>** elements defining the observed parameters and the values likely to activate reconfigurations. **<Condition>** elements allow association of reconfigurations actions (**<Action>** elements) with **<Event>** elements using Boolean expressions. In the current APSL version, we distinguish three kinds of reconfiguration actions:
    - actions such as increase, decrease, set, divide, etc. which are applied to APSL element attributes
    - actions which may affect *ilink* and *olink* attributes. These actions allow modifying the adaptation graph by modifying its edges.
    - Finally, a third kind of actions may be applied on reconfiguration policies themselves in order to validate or to invalidate a reconfiguration policy, or to change some of its attributes.

```
<APSL>
 <Network>
   <Input id="IN" src="udp://magnesium.inria.fr:5008"/>
   <Output id="OUT" src="rtp://einsteinium.inria.fr:5012"/>
 </Network>
 <Process>
   <Decoder id="D" ilink="IN" fmt="26"/>
   <Resizer id="R" ilink="D" height="144" width="176"/>
   <Encoder id="E" ilink="R" olink="OUT" frame-rate="25" quality="80" fmt="32"/>
</Process>
 <Reconfiguration>
  <Rule id="R1">
   <Probe element="OUT" >
     <Event parameter="packet-loss" id="EVT1" op="exceeds" value="10" unit="%"/>
     <Event parameter="packet-loss" id="EVT2" op="exceeds" value="50" unit="%"/>
   </Probe>
   <Condition id="C1" events="EVT1" priority="1" status="active">
     <Action name="decrease" ref="id(E)@quality" value="5"/>
   </Condition>
   <Condition id="C2" events="EVT2" priority="2" status="active" >
    < Action name="set" ref="id(E)@fmt" value="31" />
   </Condition>
  </Rule>
 </Reconfiguration>
</APSL>
```

Figure 3. An APSL specification of a video transcoding process

Figure 3 shows a simple APSL specification of a video transcoding proxy. The adaptation process consists in receiving an UDP video stream encoded in MJPEG, resizing video frames into QCIF size (176*144), encoding the result in MPEG and sending it via an RTP connection. **<Input>** and **<Output>** define respectively the MJPEG incoming stream and the outgoing adapted MPEG stream.

Reconfiguration rule *R1* introduces a probe, associated with output element *OUT,* with two events: ”packet-loss exceeds 10%” (*EVT1*) and ”packet-loss exceeds 50%” (*EVT2*). Two conditions (*C1*) and (*C2*) are respectively associated with events’ definitions. The first decreases the quality value of the encoder element. The second action changes the encoding format (*fmt* attribute) to H.261 (given by a standard payload number [19]).

3.2.2. Translation Process

The translation of APSL descriptions is made by instantiating the appropriate component architecture. According to the previous model, components in the execution level are of three kinds: *media filters*, *probes* and *actuators*.

- Media filters are basic DirectShow filters encapsulating usual multimedia functions such as: video encoding/decoding, spatial transformations, local as well as remote data streaming. It includes 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. Processing components integrate media-related functions for processing audio and video data. Examples of such components are decoders, resolution-scalar, audio and video mixers, etc.

- Probes are responsible for providing monitoring information on both application (the adaptation process) and system resource states. In the former, *QoS Probes* interact with media filters in order to gather quality parameters measured by filters (e.g. packet-loss measured by an RTP Transmitter). In the later, a set of *Resource Probes* are responsible for gathering various metrics reflecting resource states[2]. Both QoS and Resource probes can be solicited to notify significant changes in parameters it manages, and thus, triggering reconfiguration actions.

- Actuators are responsible for executing reconfiguration actions upon event notifications. The execution of reconfigurations is detailed in the next section.

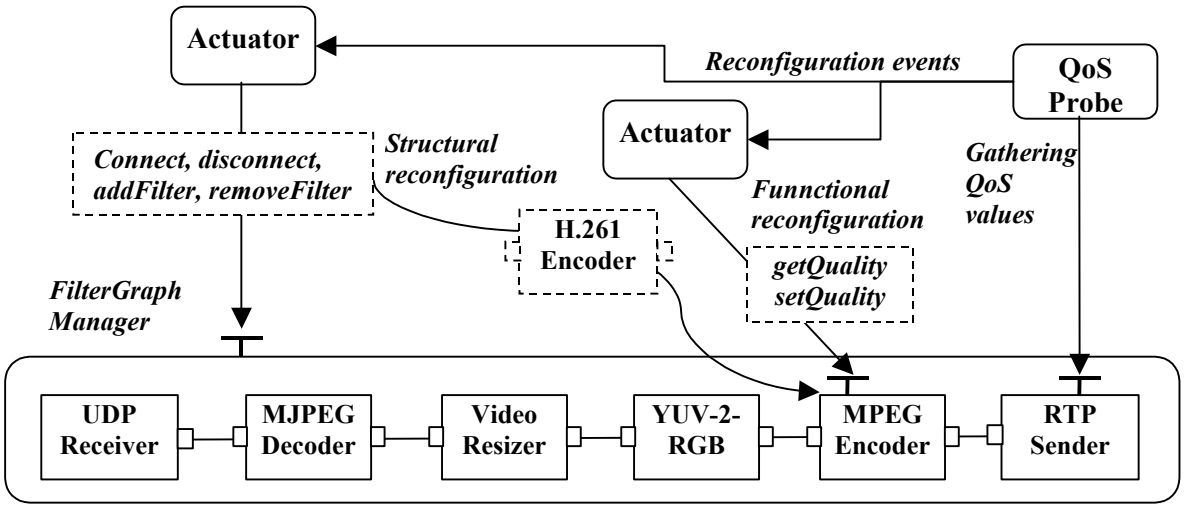

Figure 4. An example of Self-Adaptive process.

The translation process instantiates a component-based architecture from an APSL specification. Each ASPL element is translated into one (or several, see below) media filter(s) which can be seen as

[2] Our framework can be used with a monitoring probe which reports resource usage on the client device. The adaptation process can be viewed as an extension to the client application (on the device). The client device cooperates with its adaptation process by providing it with monitoring information. In this case, the adaptation process on the proxy is not transparent to the client, but it is a means to move computing load from the client to the proxy.

a refinement of its functional behaviour. Figure 4 shows the software architecture resulting from the translation of the APSL specification given in Figure 3. The <Input>, <Output>, <Decoder>, <Encoder> and <Resizer> APSL elements are respectively implemented by components: UDP Receiver, RTP Sender, MJPEG Decoder, MPEG Encoder and Video Resizer. However, our translation process may introduce additional components in order to deal with media type mismatch. In Figure 4, we assume that component MJPEG Decoder produces video frames in YUV[3] format and that component MPEG Encoder consumes video frames in RGB format. Then, a YUV-2-RGB component is introduced to convert YUV frames to RGB frames.

In a similar manner, Probes and Actuators components are created for observation and reconfiguration tasks. The next step of the translation process consists in connecting these components through their *Pins* according to their described relationships.

3.3. Reconfiguration

Applications being built-up from a set of components[4], reconfiguration operations can be achieved in two ways: functional or structural reconfiguration. Below, we detail the execution of both of them.

Functional Reconfiguration: The most basic form of reconfiguration consists in changing the functional properties of components belonging to the application. Based on this, reconfigurations may target key attributes in order to tune the media stream adaptation. Although this operation only affects one media component (a DirectShow media filter), its impact on the media type produced by this component defines two cases:

- In a first case, the modification of this attribute does not affect the produced media type and therefore, this operation does not interrupt the execution of the application (e.g. Action *Decrease* in Figure 3).
- In a second case, the modification of this attribute affects the produced media type, agreed during initial bindings of this component. (e.g. a modification of the video resolution attributes of a *Video Resizer* component). This operation requires to unbind and re-bind involved components.

Structural Reconfiguration: The second form of reconfiguration concerns the modification of the component architecture (adding or replacing a component). In general, these operations involve several steps and requires manipulation of components and their bindings. Figure 5 shows the major steps:

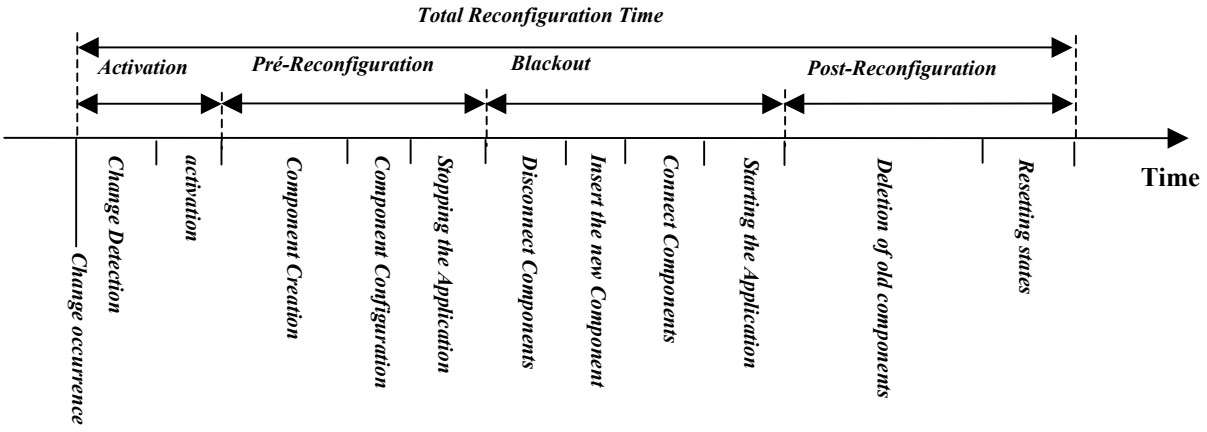

Figure 5. Steps of Structural Reconfiguration.

[3] YUV appears to be better suited for resizing video frames.
[4] APSL elements at the language level, translated into components at the execution level.

- The *Activation* step includes detection of change and decision of the activation of reconfiguration operations.
- The *Pre-reconfiguration* step encompasses all tasks that can be performed without application interruption, among them: creating new components and setting their attributes.
- The *Blackout* step represents the period during which the application is stopped. This is required to disconnect, connect, add or remove components in the application.
- The *Post-reconfiguration* step encompasses all tasks that can be made after application restarting, among them: deleting old components and resetting key attributes.

4. Testbed Application and Performance Application

The framework has been entirely implemented in C++ under Microsoft Windows 2000 using the Microsoft .Net 2003 development platform. Multimedia processing functionalities have been developed using the DirectX Software Development Kit (SDK) version 9.0.

As testbed application, we have designed SPIDER, a media transcoding gateway for adaptive video streaming over the web. Figure 6 shows the architecture of SPIDER. Several multimedia services are made available in the network, among them: a TV broadcasting service, a radio broadcasting service and a Video On Demand server providing some audio-visual content. Each of them employs specific transmission protocols and data encoding formats. Some intermediate nodes (proxies) run SPIDER, which role is to adapt video streams for each terminal capability. SPIDER relies on our framework, but it also provides a means for client applications to specify the desired behaviour on the proxy. Therefore SPIDER offers *receiver-driven* adaptive media streaming for mobile devices. The definition of the adaptation process is embedded within the client request and conveyed to the proxy using traditional session protocols (in our experiments we used HTTP).

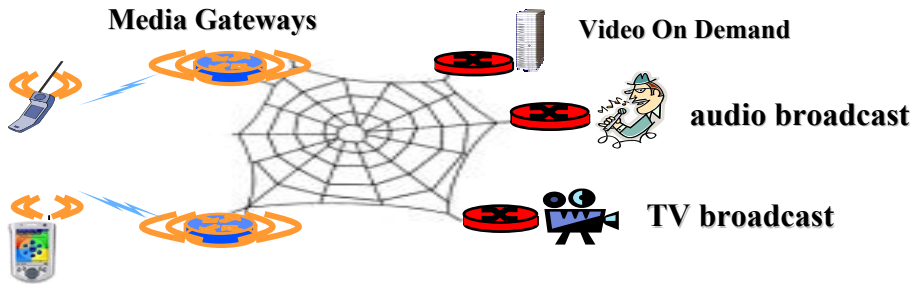

Figure 6. Experimentation Architecture.

In our experiments and evaluations, we used the following scenario. The client application starts streaming a video from a Video On Demand server. This video is encoded in MPEG with a resolution at 320x240. Displaying this video without SPIDER performs poorly because the application receives a high amount of data and must resize video frames in order to fit its screen size, thus overloading the CPU of the device.

Using SPIDER, the definition of the adaptation process specifies that if the client device is overloaded, the size of the video should be scaled down by 5% (this rule is evaluated periodically every 10 seconds). This simple reconfiguration policy allows to evaluate (and measure) the two types of reconfiguration described in section 3.3:

Structural reconfiguration. The first time a reconfiguration occurs, the component required to resize video frames (*Resizer*) is not present in the graph of the adaptation process and it has to be added, thus requiring a structural reconfiguration.

Functional reconfiguration. Subsequent reconfigurations do not have to add this Resizer component, but they just have to modify its properties, thus requiring a functional reconfiguration. This functional reconfiguration requires to unbind and rebind some of the other components.

We have implemented this scenario for our performance evaluation. As hardware setup, we used for the proxy a PC running Windows 2000 Operating System and equipped with a 2 GHz Pentium 4 processor, 256 MB of Memory and 100 MB/s LAN. On the client side, we used a Compaq IPaq H.3970 PDA running PocketPC 2002 Operating System, and equipped with a XScale Processor at 400 MHz, 64 MB of memory and 11 MB/s 802.11b WLAN. Our evaluation concerns the measurements of reconfiguration costs and their impact on both QoS and resource usage. All time measurements were based on High Resolution Performance Counters provided by Win32 API. On our hardware configuration, it provides high resolution time stamps with 100 nano-seconds accuracy.

4.1. Measurements of Structural Reconfiguration

The first occurrence of reconfiguration requires a structural reconfiguration with the insertion of a Video Resizer component (to allow resizing the video frames). Table 1 shows the mean time of this operation as well as the detailed costs of the three last steps, i.e. pre-reconfiguration, blackout and post-reconfiguration (see Figure 5). The whole reconfiguration time is about 35600 µs (micro-seconds). Most of this time is spent in the pre-reconfiguration step, which takes 24000 µs. This is explained by the instantiation of the new component, which requires loading a dynamic library and creating a new component instance. The blackout time is 11600 µs spent for insertion of the Resizer component and its connection with other components. Post-reconfiguration operations take about 2030 µs. These results show that performing pre-reconfiguration (before actually reconfiguring) significantly minimizes the cost of structural reconfiguration. Indeed, a naïve implementation would have required application be stopped during the whole reconfiguration time.

Reconfiguration Steps	**Component**		**Connections**	**Total**
	instantiation	**configuration**		
Pre-reconfiguration	23900 µs	100 µs	-	24000 µs
Blackout	11060 µs		540 µs	11600 µs
Post-reconfiguration	200 µs		-	200 µs
Total reconfiguration	35060 µs		540 µs	35600 µs

Table 1. Distribution of Structural Reconfiguration Time.

4.1.1. Measurements of Functional Reconfiguration

Next reconfigurations are functional reconfigurations which modify the Video Resizer properties. As these reconfigurations affect the output media type (the resolution), it requires the reconnection (unbind, property modification and rebind) of components that appear after the *Resizer* in the processing graph. This takes most of time, about 240 µs. Despite this, the whole reconfiguration time is kept low, about 400 µs.

4.2. Impact of Reconfiguration on QoS and Resource Usage

In order to evaluate the impact of this reconfiguration policy on both QoS and resource usage, we measured the rate of video display (in frames/seconds) and the CPU load on the client device. Figure 7 reports these measurements during the whole scenario. Vertical dotted lines highlight the occurrence of a reconfiguration. These results show that reconfigurations do not degrade performance since the reconfiguration blackout time is almost transparent to the client application. In the first time segment, video is displayed at less than 10 fps with a CPU load at 100 %. This is due to the fact that the application receives a high amount of data and in addition, it must resize video frames in order to fit its screen size. This operation requires additional processing causing the application to drop frames in

response to CPU overload. The first reconfiguration reduces the video resolution to 304x228 and therefore, increases the frame rate to 17 fps. However, the CPU load is kept at 100 % as the transmission rate remains high. Finally, the resolution is reduced to 273x205, resulting in a frame rate at 25 fps and a CPU load at 70 %. These results clearly illustrate the benefits of reconfigurations on both QoS and resource usage.

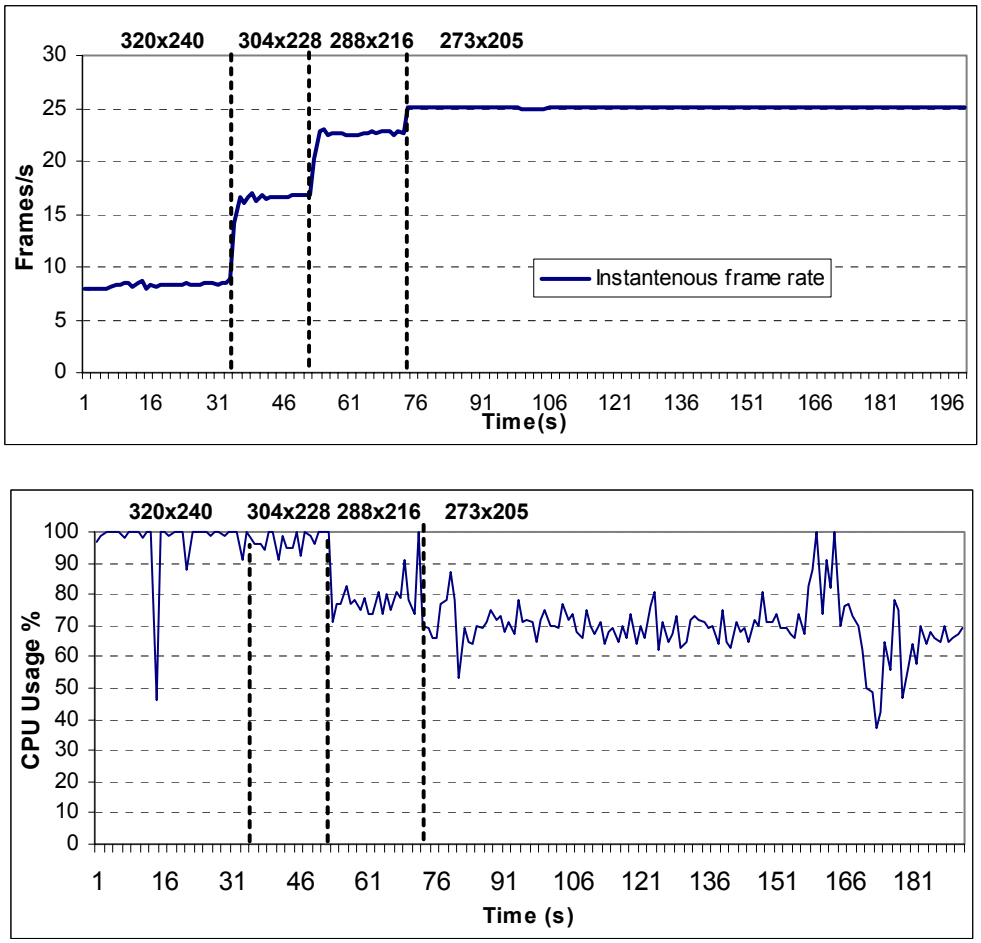

Figure 7. Impact of Reconfiguration on QoS and Resource Usage.

5. Conclusion and Future Work

This paper has described a framework for building network-based media adaptation. The main consideration concerned the dynamic configuration and the reconfiguration of adaptation processes. This framework uses a high-level specification language of both adaptation processes and their reconfiguration policies. The dynamic configuration is achieved by composing separately defined components. The reconfiguration can be performed at a component-level by modifying its properties or at the application-level by adding/removing components in the initial configuration. The experimental evaluation has shown the effectiveness of our approach and impact of reconfigurations on perceived Qos.

The lessons learned from these experiments can be summarized as follows:

- Proxy-based stream adaptation allows managing QoS in constrained execution environments transparently to servers and clients. In particular, it can be used to lighten the computing load on the client (especially when it is a mobile device).

- Adaptations are intrinsically dynamic. They have to take into account the environment constraints at launch time and at run time. Therefore, they should be self-reconfigurable.
- A component-based framework provides the required features to implement dynamic configuration and reconfiguration. A DirectShow implementation of such a framework provides a good tradeoff between flexibility and performance.
- A high level language such as APSL provides the flexibility required to face the wide spectrum of adaptations which may have to be defined to manage QoS in today's environments.

Our future work concerns mainly the distribution of an adaptation process across multiple gateways distributed over the network. Effectively, for scalability issues, distributing adaptation on multiple gateways may be required in many application scenarios. This should involve the integration of many new aspects of reconfiguration policies such as fault tolerance, error recovery, external reconfiguration events, etc.

6. References

[1] E. Amir, S. McCanne, and R. Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In Proc. of ACM SIGCOMM '98, Vancouver, Canada, August 1998.

[2] A.P. Black, J. Huang, R. Koster, J. Walpole and Calton Pu. Infopipes: an Abstraction for Multimedia Streaming. In Multimedia Systems (special issue on Multimedia Middleware), 2002.

[3] G. Blair, L. Blair, V. Issarny, P. Tuma, A. Zarras. The Role of Software Architecture in Constraining Adaptation in Component-based Middleware Platforms. In Proceedings of Middleware 2000, April 2000, Hudson River Valley (NY), USA.

[4] T. Brayand al. Extensible Markup Language (XML) 1.0. Recommendation, W3C, February 1998.

[5] D. Duke and I. Herman. A Standard for Multimedia Middleware. In ACM International Conference on Multimedia, 1998.

[6] T. Fitzpatrick and J. Gallop and G. Blair and C. Cooper and G. Coulson and D. Duce and I. Johnson, Design and Application of TOAST: An Adaptive Distributed Multimedia Middleware. International Workshop on Interactive Distributed Multimedia Systems, 2001.

[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, (Adapting to Network and Client Variability), Vol. 5 No. 4, August 1998.

[8] X. Fu, W. Shi, A. Akkerman, V. Karamcheti. CANS: Composable, adaptive network services infrastructure, in Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS'01), March 2001.

[9] V. Hardman, A. Sasse, M. Handley and A. Watson, Reliable Audio for Use over the Internet, In Proc. of INET'95, Honolulu, Hawaii, June 1995.

[10] M. Hemy, U. Hengartner, P. Steenkiste, and T. Gross. MPEG system Streams in Best-Effort Networks. PacketVideo Workshop, Cagliari, Italy, 1999.

[11] M. Johanson, An RTP to HTTP video gateway. In proc of the World Wide Web Conference 2001.

[12] O. Layaïda, S. Atallah, D. Hagimont. Reconfiguration-based QoS Management in Multimedia Streaming Applications. In Proceedings of the 30th IEEE/Euromicro Conference, Track on Multimedia and Telecommunications: Challenges in Distributed Multimedia Systems, Rennes, France, August 31st-Septembre3rd, 2004.

[13] Z. Lei and N.D. Georganas, H.263 Video Transcoding for Spatial Resolution Downscaling, In Proc. of IEEE International Conference on Information Technology: Coding and Computing (ITCC) 2002, Las Vegas, April 2002.

[14] S. McCanne and V. Jacobson. VIC: A flexible framework for packet video. In Proc. of ACM Multimedia'95, November 1995.

[15] Microsoft: DirectShow 9.0 Architecture and SDK. http://msdn.microsoft.com/directx/

[16] Z. Morley Mao, H.Wilson So, B. Kang, and R.H. Katz. Network Support for Mobile Multimedia using a Self-adaptive Distributed Proxy, 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV-2001), Port Jefferson, New York, June 2001.

[17] K. Singh, G. Nair, H. Schulzrinne. Centralized Conferencing using SIP. In Internet Telephony Workshop IPTel'2001.

[18] H. Schulzrinne and al. RTP: A Transport Protocol for Real Time Applications. RFC 1889.

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

[20] Sun: Java Media Framework API Guide. http://java.sun.com/products/javamedia/jmf/

[21] 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. 1999.

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

[23] World Wide Web Consortium Recommendation. *World Wide Web Consortium Recommendation, Transactions on Communications, Vol. E82-B.* http://www.w3.org/TR/xpath.html Nov. 1999.