Adaptive Media Streaming Using Self-reconfigurable Proxies

Oussama Layaida, Slim Benattalah and Daniel Hagimont

SARDES Project, INRIA Rhône-Alpes
ZIRST-655, Avenue de l'Europe- 38334 Montbonnot Saint-Ismier Cedex, France.
Phone : +33 4 76 61 52 79 - Fax: +33 4 76 61 52 52
Oussama.Layaida@inrialpes.fr

Abstract. With the increasing number of networked devices, multimedia applications require additional functionality inside the network in order to adapt multimedia streams. In this field, much research work has been proposed; however, the dynamic configuration and the reconfiguration at run-time of such services remain little studied. This paper addresses these issues and describes a framework for the dynamic configuration and the reconfiguration of network-based media adaptation. We show through experimental evaluations that besides adaptations, reconfiguration can improve significantly the performance of client applications with a minimal cost.
Keywords: Media Streaming, Proxy, Qos, Reconfiguration.

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. On the other hand, multimedia applications evolve in dynamic environments with 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 life cycle.

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 engines inside

the network in order to adapt the multimedia content according to network, hardware or software capacities [6] (called Proxy or Network-based adaptation). An adaptation process may consist in filtering multimedia streams [8], changing their transmission protocol [10], transcoding content [1] or applying spatial and temporal transformations [20][12]. The major benefit of network-based adaptation is its non-intrusivity with respect to legacy applications, while profiting of the processing power available in the data-path. Adaptation tasks are performed by the proxy transparently to servers and clients. Neither servers have to change its transmission technique, nor clients have to take care about the adaptation process. 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 underlying environment, an adaptation process must change its behavior 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 details the architecture of the framework and its major components. Section 3 gives a performance evaluation with different application scenarios. Thereafter, section 4 gives a discussion and a comparison with related work. Finally, we conclude and present future work in section 5.

2 Framework Architecture

This section describes our framework. It starts with an overview of our approach and then details the framework architecture through an application scenario.

2.1 General overview

To address the previous requirements, our approach is based on the separation of the specification of the adaptation process from its implementation. The

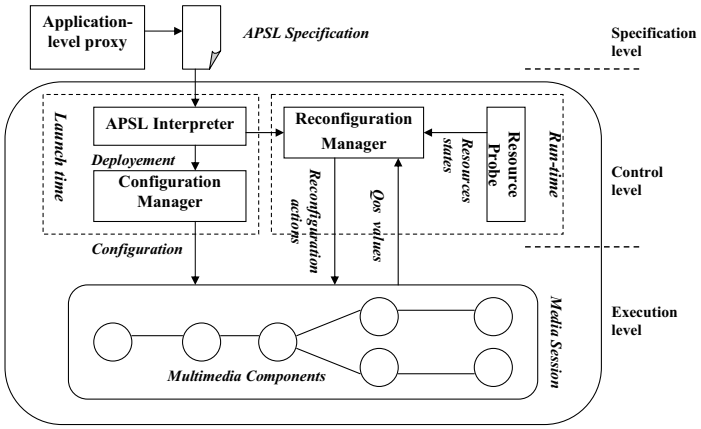

Fig. 1. General overview of the framework.

specification of an adaptation process includes its configuration (i.e. structure) and reconfiguration policies ensuring its evolution. Specifications are then interpreted and translated into a run-time adaptation process. To achieve this goal, we employ a component-based platform where basic functions are implemented in separately defined components. The dynamic configuration of adaptation processes is performed by assembling the required set of components. The reconfiguration is performed by modifying either parameters of a given component, or the structure of the component configuration. These are the essential features in order to address the large scope of adaptation and reconfiguration requirements without additional development effort. As depicted in figure 1, the framework is structured in three levels:

- The specification level an XML-based [3] specification language named APSL (Adaptation Proxy Specification Language).
- The control level integrates all required functions to translate APSL specifications into self-reconfigurable adaptation processes.
- The execution level includes basic components from which adaptation processes are built.

2.2 The APSL Language

The aim of APSL is to provide a general mean for the specification of media adaptation processes. These last are specified as a set of basic multimedia related tasks such as, network input/output, decoding, encoding, etc. Each of them is expressed with a specific element and a collection of attributes that precise its properties. These tasks are bound in an acyclic graph using binding attributes (ilink and olink). Figure 2 shows a simple APSL specification of a video transcoding proxy. It consists in receiving an UDP video stream encoded

in MJPEG, resizing video frames into QCIF size (176*144), encoding in MPEG and sending the result using RTP. Besides the description of adaptation configuration, APSL offers constructs for the specification of reconfiguration policies that define how it should react to changes. Reconfiguration policies are expressed in the form of probes, conditions and actions. Each probe is composed of one or more *Events* defining the observed parameters and the values likely to activate reconfigurations. A *Condition* is used to associate reconfigurations actions with events. It is possible to compose several events within the same condition using boolean expressions in *evt* attribute (see figure 2). Reconfiguration actions define arithmetic operations (increase, decrease, set, divide, etc.) on attribute values (addressed using XPath expressions [21] in *evt* attribute). Depending on the type of the targeted attribute, we distinguish three kinds of actions:

- First, actions are applied on functional parameters of the adaptation process such as encoding quality, frame rate, resolution, etc.
- In a second case, actions may manipulate the structure of the processing graph by modifying links between processing elements.
- Finally, a third kind of actions may be applied on reconfiguration policies themselves in order to (in)validate a policy or to change its behavior (event thresholds, arithmetic operations, etc.).

Our example defines a probe attached to the network output element with two events: (1) packet-loss exceeds 10 % and (2) packet-loss exceeds 50 %. Two conditions associate actions with those events. The first consists in decreasing 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, see [17]).

2.3 Multimedia Components

Multimedia components are the basic building blocs from which adaptation processes are built. We have chosen to implement multimedia components using the DirectShow model [4]. A DirectShow component encapsulates an atomic operation on multimedia data such as, encoding, decoding, etc. It is characterized by one or more input stream interfaces (called *Input pins*) used to receive data in a specific media type, and one or more output stream (*Output pins*) interfaces for delivering transformed data in a specific data type. 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. In addition to stream interfaces, a component exposes one or more configuration interfaces that allow setting-up the functional properties tuning its behavior.

In our framework, there are 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. Processing components integrate media-related functions for processing audio and video data. Examples of such components are decoders, frame dropper, resolution resizer, audio and video mixers, etc.

```
<APSL>
  <Network>
      <Input id="IN" src="udp://magnesium.inrialpes.fr:5008" />
      <Output id="OUT" src="rtp://einsteinium.inrialpes.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(E1)@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>
```

Fig. 2. An APSL specification of a video transcoding process.

2.4 Configuration Manager

The *Configuration Manager* offers the required functions in order to build adaptation processes as a composition of multimedia components. This operation is performed as follows:

1. First, the APSL parser syntactically parses specifications and checks their validity in terms of media type compatibility between elements and graph correctness.
2. A composite component, named *Media Session*, is created first to own basic multimedia components. As shown on figure 3, this component provides the IMediaSession[1] interface to manipulate its sub-components, their bindings and the execution of the whole structure (run, stop, pause).
3. For each APSL processing element, the required multimedia components are created and configured with information extracted from the APSL specification. APSL information is maintained within components to represent their current configuration. As we will see, such information is used to retrieve the appropriate components during reconfiguration tasks.
4. Components are connected in the media session according to bindings in the APSL specification. During this step, other components, not explicitly declared in the APSL specification, may be inserted to coordinate connections between processing components. Examples of such components are: media type converters, stream duplicators, multiplexers, de-multiplexers, etc.
5. The resulting media session is started.

[1] We use the prefix 'I' to indicate interface names.

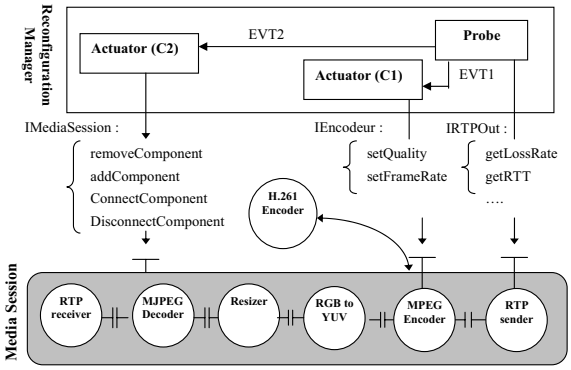

Fig. 3. Configuration and reconfiguration of an adaptation process.

To clarify this process, we show on figure 3 the media session created from the specification on 2. First, an RTP receiver and an MJPEG decoder are used to receive the original stream from the server and uncompress its content. A Resizer component transforms the video resolution into QCIF. Then, an MPEG encoder and a RTP sender produce the adapted stream. The RGB-to-YUV converter is inserted to overcome a media type mismatch between the Resizer and the MPEG decoder.

2.5 Reconfiguration Manager

The *Reconfiguration Manager* is responsible for the execution of reconfiguration policies within the media-session and its subcomponents. During the configuration step, it creates `Probe` and `Actuator` components (see figure 3). A Probe corresponds to the Probe in the APSL specification and is responsible for triggering reconfiguration events. As shown in figure 1, we distinguish between Resource and Qos Probes. Resource Probes act as separate monitors and notify significant changes in resource states such as CPU load, memory consumption, remaining battery life-time, etc. Qos probes interact with processing components to return events related to quality of service (Qos) parameters (e.g. packet loss, transmission rate, etc.). Such events are notified to Actuators, which are responsible for the execution of reconfigurations actions. Note that each of Probes and Actuators determines its behavior and media components with which it interacts with the help of APSL information stored in each component. In our example, the involved components are the RTP sender (OUT), the encoder (E) and the media session. The Probe uses the `IRTPOut` interface of the RTP sender component to periodically monitor the packet loss rate experienced during the RTP transmission. It may notify two events: the first (EVT1) occurs when packet loss exceeds 10 % and the second (EVT2) when it exceeds 50 %. Such events are reported to the corresponding Actuator, which performs the appropriate reconfiguration actions. We distinguish two kinds of reconfigurations:

– *Parameter Reconfiguration:* We call *parameter reconfiguration* the modification of some key properties of a component participating in the adaptation process. Actuator C1 (figure 3) performs such an operation for decreasing the encoding quality factor. This is achieved using the `IEncoder` interface of the encoder component (`getQuality` and `setQuality` methods).
– *Structural Reconfiguration:* Media sessions being built as component-based architectures, the second kind of reconfiguration consists in modifying the structure of the adaptation process. Such reconfigurations are performed by dynamically adding, removing, connecting and disconnecting components. Actuator C2 performs a similar operation in order to replace the MPEG encoder component by an H.261 encoder component. This reconfiguration is achieved through the IMediaSession interface by: (1) creating an H.261 encoder (2) disconnecting and removing the MPEG encoder and (3) adding and connecting the H.261 encoder.

3 Experimental Evaluation

The framework has been entirely implemented in C++ under Microsoft Windows 2000 using Microsoft .Net 2003 development platform. Multimedia components have been developed using DirectX Software Development Kit (SDK) version 9.0. Several application scenarios have been built and the obtained results have shown the benefits of adaptations on client performance, especially for terminals with limited resources [2]. Below, we report an experimental evaluation of our framework. In all experiments, we used a Windows 2000-based PC with a Pentium 4 Processor at 2 Ghz and 256 Mo of memory to play the role of an adaptation proxy (or server). As client terminals, we used mobile PDAs (Compaq IPaq H3970) equipped with a PXA250 Processor at 400 Mhz and 64 Mb of memory, running Pocket PC 2003 operating system. Those terminals are connected to the Internet through a 2 Mbps 802.11b wireless network.

3.1 Configuring media sessions

First, we evaluate the configuration process of multimedia sessions. For this goal, we performed experiments with 5 application scenarios built from APSL specifications: (1) A multicast-to-unicast RTP gateway, (2) an HTTP-to-RTP video gateway, (3) an H.261 to H.261 video transcoding proxy with resolution downscaling, (4) an H.261 to H.261 video transcoding proxy with two clients and (5) a conferencing server mixing video streams between 4 participants. Evaluation criterias are the deployment time and the CPU consumption at run-time. The deployment time includes operation of parsing APSL specification and configuration tasks till launching the application. It is measured using high resolution performance counters provided in Win32 API. The CPU load is measured with one application running at a time.

Figure 4.(a) shows the deployment time according to the number of components involved in each application. It varies between 30 ms for 2 components

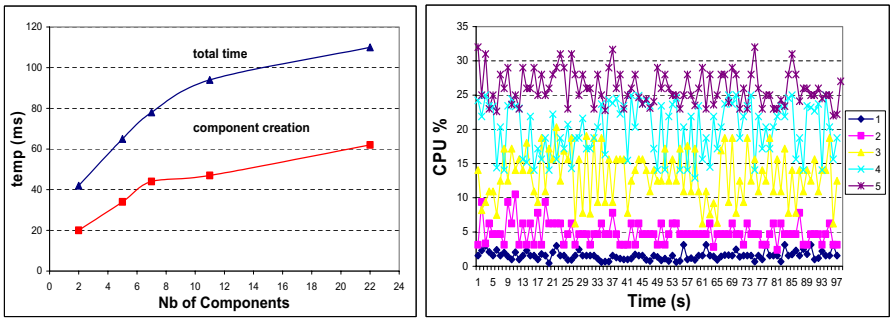

Fig. 4. (a) Configuration time (b) CPU load.

(scenario 1) to 110 ms with 22 (scenario 5). Regarding to scenarios 1, 2 and 3, the configuration time increases linearly with the number of multimedia components (with an average of 15 ms per component). However, this 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. (e.g. the videoconferencing server creates RTP senders and receivers for each participant). Indeed, as multimedia components are hosted in dynamic link libraries, creating an instance of a component requires first loading its DLL, which is also used for other instances. In practice, several multimedia sessions are created at the same time which increases sharing DLLs. At the other hand, the CPU load, given in figure 4, varies with the nature of the treatment performed in the media session. It varies between 3 % for the first configuration to 28 % for the videoconferencing server. These results seem to be acceptable as all applications were successfully built and executed.

3.2 Impact of adaptation on Qos

In order to evaluate the effects of adaptations on client performances, we measured the Qos perceives at the client side within the video streaming scenario (scenario 3). As a client application we use VVP [19], a videoconferencing client application for Pocket PC. Our evaluation consists in comparing two cases. In the first, the client application receives the original stream encoded in H.261 and CIF resolution (352*288). In the second case, the client receives video content through the proxy which reduces the resolution to QCIF (176*144). We measured in both cases the displayed frame rate and the data receiving rate, reported respectively on Figure 5.a and 5.b. With the first configuration the data bit rate is around 900 Kbps with all streams. The frame rate of the displayed stream is lower than 10 frames per second. In the second configuration, the data bit rate is around 200 kbps as the client receives only one stream. As a result, the frame rate reaches the original frame rate of 25 frames/second.

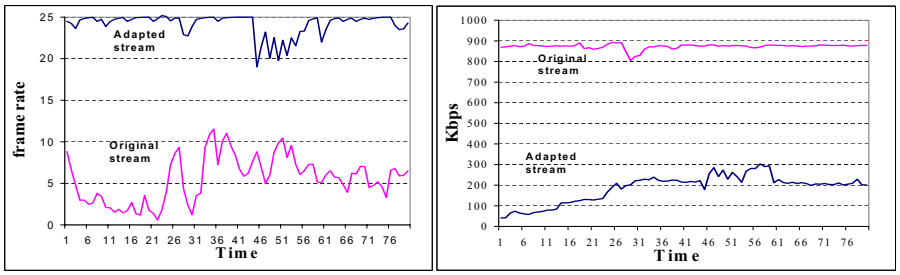

Fig. 5. Impact of adaptation: (a) displaying frame rate (b) data bit rate

3.3 Reconfiguring media sessions

The second evaluation concerns reconfiguration tasks. We have performed our experiment within the application illustrated in figure 3. The reconfiguration action consists in replacing the encoder component at runtime. The average time to perform this operation was about 43 milliseconds. The instantiation and setting-up of the H.261 encoder component are performed before interrupting the application in order to minimize interruption time. This task takes most of the reconfiguration time ($\simeq$ 28 ms). The application was paused during the remaining time ($\simeq$ 15 ms) in order to replace the encoder component. Knowing that the configuration of an equivalent session takes about 70 ms (scenario 3), these results illustrate clearly the benefits of reconfiguration.

3.4 Impact of reconfiguration on Qos

We now evaluate the impact of reconfiguration on the perceived Qos at the client side. With the previous experiments (section 3.3, we use an RTP-based client application supporting both MPEG and H.261. We measure the displayed frame rate (in frames/seconds) and the packet loss rate observed by the application during the transmission. Results are reported on figure 6. We distinguish three different stages: (i) before (ii) during and (iii) after the reconfiguration:

- Before the reconfiguration process, the packet loss rate was around 30 % and the frame rate under 20 fps. Loss rate increases until it exceeds 50 % and causes the reconfiguration.
- During the reconfiguration, the frame rate decreases down to 0 fps, inducing a blackout time of 500 ms. Although the proxy has been paused during only 15 ms, this time is spend by the client to detect change of the encoding format in the RTP stream (RTP fmt and payload-specific headers, see [16, 17]) and create a new decoder.
- After the reconfiguration has taken place, the client application starts displaying video at 25 fps. The loss rate is kept under 10 % as the new stream

meets terminal's capacities. This shows that, besides adaptations, reconfiguration improves significantly client performances for the price of a short blackout time depending of the client application behavior.

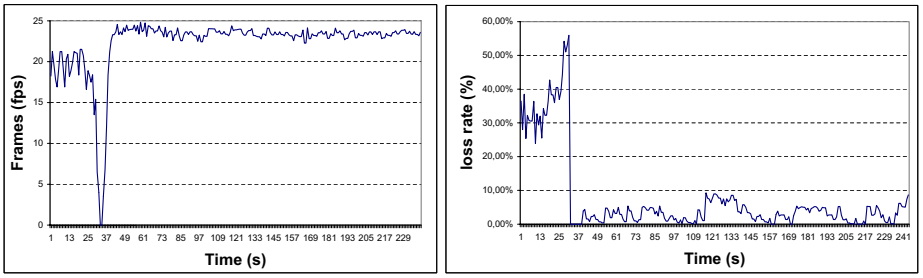

Fig. 6. Impact of reconfiguration: (a) Displaying frame rate (b) packet loss rate.

4 Related Work

Network-based solution has demonstrated the potential benefits of using the processing power available on the data path to perform adaptation tasks. This advantage has motivated numerous works such as [8, 20, 10, 18, 12]. Although these proposals have considered some heterogeneity problems, they have addressed specific class of adaptation and focused on algorithms for data stream transformations, congestion control, optimizing data transcoding, etc. The dynamic configuration and reconfiguration of adaptation processes were beyond their scope.

Some research efforts around adaptive multimedia applications addressed issues of the dynamic configuration of network-based adaptation. 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. Infopipes [11] defines an application-level framework that allows configuring adaptation processes by composing separately defined components. The limitation of these works is that they do not consider changes in the environment and therefore, do not provide support for reconfiguration. With a more general purpose, some component-based platforms provide support for developing reconfigurable multimedia applications. TOAST [5] (Toolkit for Open Adaptive Streaming Technology) explores the use of open implementation and reflection to build reconfigurable multimedia applications. Reconfiguration methods offered in TOAST, similar to ours, can be addressed at different levels. However, the lack of a specification tool left reconfiguration issues to application developers.

In contrast to this approach, our work aims at offering a general mean to build reconfigurable adaptation processes without additional development efforts.

Closely related to our work, CANS (Composable Adaptive Network Services) [7] addresses the dynamic configuration of network-based adaptation as well as their reconfiguration. Reconfiguration tasks are performed by a set of managers responsible for monitoring resources and applying reconfigurations when needed. However, reconfiguration policies are hard-coded with managers and focus mainly for error recovery. This is not efficient since it does not allow customizing them according to application requirements. A similar approach was conducted in [15] for adaptive streaming in mobile multimedia applications. Other works, like [13], have followed a declarative approach for the specification of reconfiguration policies within multimedia applications. Such policies are expressed with general rules way without a definition of the application structure. This suppose that the associated parser has a perfect knowledge of the application's architecture as the specification tool does not precise which components initiates reconfiguration events and which should be targeted by reconfiguration actions. Unlike this approach, the APSL language integrates the specification of both the application and reconfiguration policies, thus addressing a broader range of media adaptation and reconfiguration requirements.

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. 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 more interesting in many application scenarios. This involves the integration of more new aspects of reconfiguration policies such as fault tolerance, error recovery, external reconfiguration events, etc.

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. S. Ben Atallah, O. Layaida, N. De-Palma, D. Hagimont. Dynamic Configuration of Multimedia Applications In Proceedings of the 6th IFIP/IEEE International Conference on Management of Multimedia Networks and Services(MMNS'03), Belfast, Northern Ireland, September 6-10, 2003
3. T. Brayand al. Extensible Markup Language (XML) 1.0. Recommendation, W3C, 1998.
4. Microsoft: DirectShow Architecture, http://msdn.microsoft.com/directx/. 2004.
5. Fitzpatrick et al. Design and Application of TOAST: an Adaptive Distributed Multimedia Middleware Platform, In Proc. 8th Intl Workshop on Interactive Distributed Multimedia Systems (IDMS01), Springer-Verlag, Lancaster, September 2001.
6. A. Fox and al. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. IEEE Personal Communications, 1998.
7. X. Fu, W. Shi, A. Akkerman, and V. Karamcheti, CANS: Composable, adaptive network services, USENIX Symposium on Internet Technologies and Systems (USITS), March 2001
8. M. Hemy and al. MPEG system Streams in Best-Effort Networks. PacketVideo Workshop 1999.
9. Sun: Java Media Framework API Guide. 2002. http://java.sun.com/products/javamedia/jmf/
10. M. Johanson, An RTP to HTTP video gateway. In proc of the World Wide Web Conference 2001.
11. 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.
12. 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, Apr. 2002.
13. B. Li and K. Nahrstedt, A Control-based Middleware Framework for Quality of Service Adaptations, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 9, September 1999.
14. R. Mohan and al. Adapting Multimedia Internet Content for Universal Access. IEEE Transactions On Multimedia, March 1999.
15. 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).
16. H. Schulzrinne and al. RTP: A Transport Protocol for Real-Time Applications. RFC 1889.
17. H. Schulzrinne. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 1890.
18. K. Singh and al. Centralized Conferencing using SIP. Internet Telephony Workshop IPTel'2001.
19. Thorson M. *VIC viewer for PocketPC.* http://www.oncoursetech.com/video/default.htm
20. 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.
21. W3C Recommendation, XML Path Language (XPath) Version 1.0. 1999.
22. X. Wang, H. Schulzerinne. Comparison of Adaptive Internet Multimedia Applications. IEICE Transactions on Communications, Jun 1999.