

# A Platform-Independent Adaptive Video Streaming Client for Mobile Devices

Andre Stephan Baumgart

Automation Laboratory  
University of Mannheim  
Mannheim, Germany  
andre.baumgart@ti.uni-mannheim.de

Hartwig Knapp, Martin Schader

Department of Information Systems III  
University of Mannheim  
Mannheim, Germany  
{knapp,schader}@wifo.uni-mannheim.de

Sebastian Mill

Chair of Software Technology  
University of Mannheim  
Mannheim, Germany  
mill@student.uni-mannheim.de

## Contact details of the first author:

Dipl.-Kfm. Andre S. Baumgart

Automation Laboratory  
University of Mannheim  
Computer Engineering  
B6, 23-29 Part B  
D-68131 Mannheim

Tel.: +49-621-181-2776

Fax : +49-621-181-2739

Email: [andre.baumgart@ti.uni-mannheim.de](mailto:andre.baumgart@ti.uni-mannheim.de)

<http://www.proaut.uni-mannheim.de/>

## ***Abstract***

The paper provides lightweight client support for disruption tolerant end-to-end optimized video streaming. The client prototype consists of two main components: first, the video player interacts with the user for explicit and implicit user-defined QoS attributes. Second, the stream load balancer adaptively handles the pre-fetching of the video stream on the client for different modes of device and connection quality parameters and notifies the video player to decode the correct video stream sequence. We show that new generation mobile phones can provide flexible video streaming capabilities with reasonable quality of service for users in a manner that is independent of the platform.

## ***Keywords***

Mobile Applications, Mobile Video Streaming, J2ME, Quality of Service

# A Platform - Independent Adaptive Video Streaming Client for Mobile Devices

Andre Stephan Baumgart  
Automation Laboratory  
University of Mannheim  
Mannheim, Germany  
andre.baumgart@ti.uni-mannheim.de

Hartwig Knapp, Martin Schader  
Department of Information Systems III  
University of Mannheim  
Mannheim, Germany  
{knapp, schader}@wifo.uni-mannheim.de

Sebastian Mill  
Chair of Software Technology  
University of Mannheim  
Mannheim, Germany  
smill@rumms.uni-mannheim.de

**Abstract** - Video streaming in mobile environments is a very complex and challenging problem. It requires the control of scarce computing resources on mobile devices, such as mobile phones, to maintain the best possible application and service performance. Adaptation can be caused by changes in workload, available resources and/or new quality of service (QoS) requirements by the user. This paper presents a flexible approach, which adapts to the different application modes triggered by the resource environment and user requirement for video streaming on mobile java enabled phones. We provide lightweight client support for disruption tolerant end-to-end optimized video streaming, which is realized as a prototype in a platform independent J2ME environment. The client prototype consists of two main components: first, the video player interacts with the user for explicit and implicit (application and device-specific features) user-defined QoS attributes. Second, the stream load balancer adaptively handles the pre-fetching of the video stream on the client for different modes of device and connection quality parameters and notifies the video player to decode the correct video stream sequence. Simulation results from our prototypical implementation show that new generation mobile phones can provide flexible video streaming capabilities with reasonable quality of service for users in a manner that is independent of the platform.

**Keywords** - Mobile Applications, Mobile Video Streaming, J2ME, Quality of Service

## I. INTRODUCTION

Video streaming on mobile devices was not very important in the past because the second generation of mobile communication systems did not fulfil the need for high data rates. With the increasing distribution of innovative technologies like UMTS in Europe, cellular providers have the possibility to offer their customers many real time applications, such as video streaming or television on demand. Heterogeneous mobile devices avert data to be simply sent from a server to a mobile client. These clients may differ in computational power, supported protocols, memory size, or display size and capabilities. Due to the fact that computational power rises above average and size decreases proportional, the heterogeneity of mobile devices will grow continuously.

There are three different methods to describe or show data on a mobile device sent by a server. First of all, one can modify every client in such a way that the devices are

enabled to read or display the data. Secondly, a redundant copy of the original data can be provided for every type of mobile device, and thirdly, one can process the data on the server and provide them in a client-readable format. Because mobile devices become more different and storage of data for every client is not an option, this alternative is not realizable. Furthermore, the growing dissimilarity of the more and more reduced devices eliminates the possibility of finding a standard for all items. Hence, our research addresses the problem of dealing with the data which are sent to different clients by a server. We will present an adaptive service that processes the data to the client and administers functions such as storing, pre-fetching, retrieving, and adapting of media content [1]. To combine the presentability with a practical example, we focused our research on video streaming applications.

In the following paragraphs, we adhere to the question of how to transport the data (stored on a server) to a mobile client and how to process and display the video data effectively to fit the constraints of the device. First, we developed a media player which works on nearly every new generation mobile device. An XML catalog file containing the video format specifications will be held on the server. Clients may download this catalog and decide to download a file via the HTTP protocol, which supports streaming technology. On the server side, we store the video files and they will be converted into a client readable video format by a subjacent layer. This layer contains an adaptive service. This service decides how to process the data to the mobile client. It will also be part of the client side and will build the key element of our development. Furthermore, we will discuss several techniques of pre-fetching of lost video frames, as mentioned in [2].

In the following section, we set the context of our work by briefly introducing the technologies we will be referring to (namely XML, HTTP, J2ME and QoS). Section III presents the architecture and components in the context of the existing technological framework. The prototypical realization shown in Section IV provides a description of our adaptive lightweight client by means of a working and evaluated prototype example. Section V subsumes our approach to other research. Concluding remarks are drawn in Section VI by giving an outlook on further research activities.

## II. SETTING THE CONTEXT

We are considering the context, in which a possible mobile device requests a video from a server. Since the server is unable to have an a priori information about the client features, such as number of colors, working resolution or size of the display, the server and the client have to exchange this relevant information. We use the HTTP protocol (as described in the next Section) to communicate because it is an existing standard which is implemented on most mobile devices of the newest generation.

Afterwards, the client requests an XML catalog stored on the server. This catalog contains information about the videos, their formats, and specifications. The client chooses one of the videos (which is listed in the XML catalog) and asks the server to send the video file using an adaptive service (will be explained in Chapter three). The protocol that is being used is, again, HTTP, due to the fact that it provides important features like video streaming functionality.

### A. Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is a protocol for distributed, collaborative, hypermedia information systems. HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems. In this way, HTTP allows basic hypermedia access to resources available from diverse applications. Furthermore, HTTP is a request/response protocol which is explained in detail in [3].

In our case, we use the HTTP protocol because of its standardized communication and streaming technology. This enables us to start transmitting streaming video data and to stop and resume transmitting on a pre-set position.

### B. Extensible Markup Language (XML)

The Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web [4]. XML describes the content in terms of what the data is that is being described and it is 'extensible' because, unlike HTML, the markup symbols are unlimited and can be user-defined.

We use XML because it is very easy to add new attributes to an XML file. In our case, an XML catalog contains all the video information and each video may have different attributes [5]. Other advantages of XML are that XML is able to store structured data in a text file, that allows syntactical and semantic validation of a document and many XML parsers are available for free.

### C. Java 2 Micro Edition (J2ME)

J2ME, which is aimed at low memory consumer devices, provides a runtime environment for applications implemented in a stripped-down version of the Java language. Applications written in this Java dialect can be used on most phones of the newest generation [6]. The implementations on the

TABLE I  
MOBILE CLIENT QoS PARAMETERS

Parameter	Description
Display Size	Number of pixels on display
Color Depth	Number of colors displayable
Memory Size	Size of storage
CPU Performance	Speed of processor
Connection Quality	Network connection quality
Video Type	Video stream type, e.g. mpeg, 3gp

mobile devices differ from the reference implementation of Sun Microsystems with the result of some incompatibility. The architecture consists of three layers: Java Virtual Machine, configuration, and profiles, which are explained in detail in [7].

### D. Quality of Service (QoS) Context

Quality of service issues have been the topic of several research studies and standardization efforts during the last years [8]. In particular, guaranteeing dependable and adaptive QoS models has become an important factor for dynamic application environments [9]. In our lightweight client approach, we need to identify the relevant QoS parameters for the video streaming application objects [8]. Server services, networks, and devices can be subject to different QoS parameters. Thereby, we focus on the client device QoS parameters presented in Table II-D. The parameters can be directly checked by the mobile device. Thus, the client itself can determine dynamic dependable adaptation to change runtime configurations with respect to individual QoS objectives [10], [11].

## III. PROTOTYPE ARCHITECTURE

This Section describes the client prototype architecture and its main components. Figure 1 summarizes the architectural and general technological context. Section IV then shows a concrete implementation of this framework.

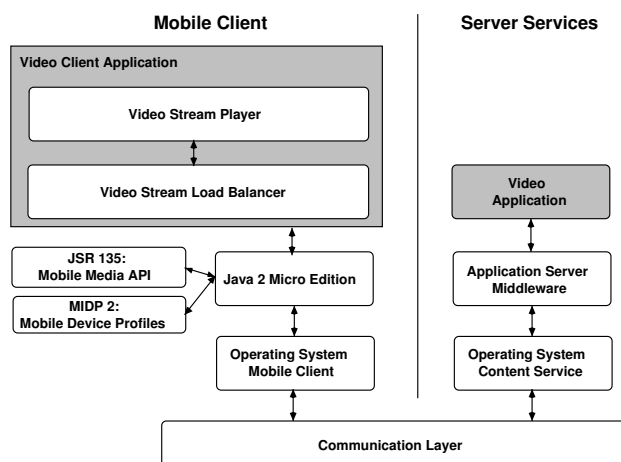


Fig. 1. Architectural Client Framework

### A. Architectural Client Framework

As mentioned in Section II, the mobile client is situated in a dynamic and heterogeneous mobile context. Various server services can be accessed by the mobile device through specialized communication channels. The client device offers specialized applications to access communication channels and service providers. Predominantly, video streaming is solved in a platform dependent manner by adjusting the mobile application to the client operating system. The Java 2 Micro Edition offers an API framework to access the functionality of the client operating system and device. The video client application uses the J2ME environment to address the needs of the application and to access the mobile device resources.

### B. Video Client Application

1) *Video Stream Player*: The video stream player is responsible for the communication between the user and the client device. Moreover, the player provides an explicit and implicit quality of service based (QoS) graphical user interface (GUI). Explicit communication allows the user to control the video(s) directly through a common interface. The user is able to select, start, stop, and resume a video. In addition, the video player implicitly allows to request video streams that do not depend on user control. Device-specific GUI parameters like resolution type or color depth are automatically sent upon a video request. Furthermore, user preferences, like a recently used or viewed video list are stored by the video player to pre-select the services offered.

2) *Stream Load Balancer*: The stream load balancer is a layer situated below the application layer on the client side, which contains the video player. This manager is responsible for session management and provides an adaptive service, which decides on the use of the offered services depending on the environment. In this chapter, we will discuss both key

features and explain current problems and their solutions by using this approach.

First of all, the client has to detect some of its properties like display size, color depth, display resolution, and memory size. Most of the newest generation mobile devices are equipped with multimedia memory cards; the storage size of these cards increases consistently. This plays an important role in our proposed solution approach and the assumed QoS requirements.

All of these properties are needed to inform the server of the specified format the client wants to receive and especially what size the data packages must not exceed to prevent a memory overflow. The server converts the source video using the client's parameters and starts sending the first part of the video file.

If the client has received sufficient data to display the first part of the file, it starts displaying the video in an independent thread. Simultaneously, a second thread is activated by the stream load balancer and starts pre-fetching the next video file segment. Depending on the bandwidth of the connection, the balancer has to check and edit the file size repeatedly to prevent stopping the displayed video on the client. If the bandwidth or the connection quality is reduced by external events the data packets will have to be reduced, as well.

Since the memory size of each mobile device is limited, our adaptive service has to start a memory size-dependent garbage collection. The service starts overwriting the video data in each thread after it has been displayed completely and the other thread has completed the pre-fetch and is ready to display its data packet. The memory size used in mobile clients increases permanently and we assume that the client may write at least 8 MB of data (for each thread) on its memory.

The user should be able to view the video file on a free selectable start position and at the last stop position respectively. Therefore, the client stores the video stream position before starting the garbage collection in a file and puts it in a memory area (e.g., in the internal device memory and not on the memory card), which will not be overwritten by the video data. This guarantees that error handling and fault recovery is done appropriately and effectively.

Generally, we have to differentiate between forced and unforced interruptions. On the one hand, the forced interruption might be a user who wants to stop viewing or abort downloading the video. The video position is then stored in a file and the user is able to resume (or restart) video streaming at any time. If the video is aborted by the user the part of the video that has been downloaded so far will be erased. While playing the video, the garbage collection starts erasing the segments of the video which have already been displayed and frees memory on the client.

On the other hand, the unforced interruption may occur if the connection to the server is interrupted, e.g. by a low or bad network connection. In this case, a fault and exception handling function stores the last received video position in a file and puts it on the client. If the user closes the application there is nothing to do until it is reopened; or the client tries to

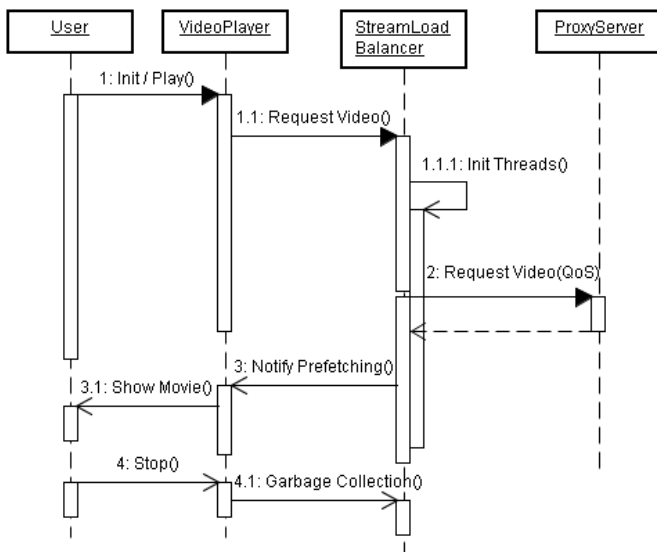


Fig. 2. Client Prototype

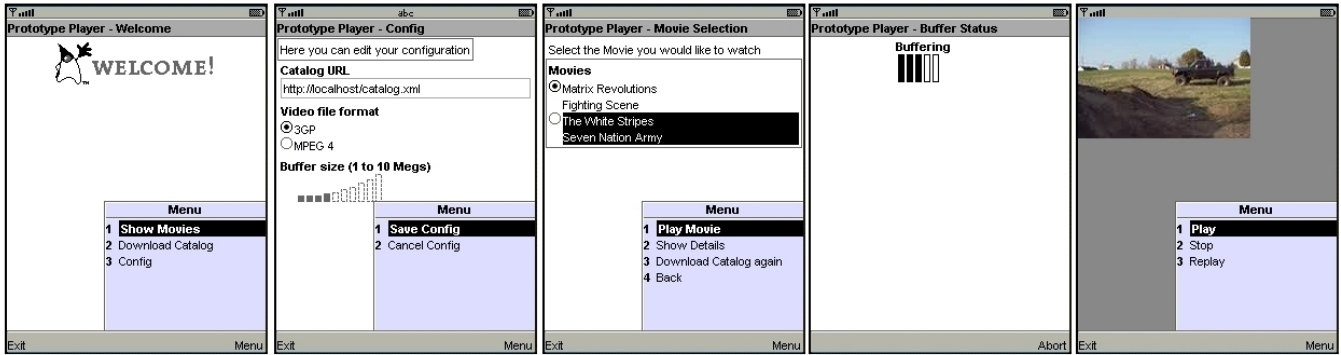


Fig. 3. Menu Sequence of the Video Player Prototype

reconnect to the server and to resume the video stream at the last stored position. In the case of an unforced interruption, the client discovers its environment again, especially the quality and bandwidth of the new connection. The client could for example be equipped with a WLAN adapter and in reach of an access point. If so, the client has to request the video data in smaller data packets because the available memory fills up more quickly.

Figure 2 shows the standard use case of a user-forced scenario in our prototypical implementation. This is presented in the next Section.

## IV. VIDEO STREAMING PROTOTYPE

### A. Prototype Environment

The prototype has been implemented for limited mobile devices (such as mobile phones) with a J2ME runtime environment. Our approach focuses on the development phase starting with a generic application development environment. This allows an appropriate adaptation and experimentation for different devices, connection, and functionality settings. Figure 3 shows the GUI widgets for the prototype video player.

The prototype has been implemented with Eclipse SDK in combination with the EclipseME plug-in, which allows us to develop our application on several devices, such as Sun's Wireless Toolkit (WTK). The WTK gives us the possibility to run and debug mobile applications on the platform-independent Java architecture. Besides that, we decided for the Nokia Prototype SDK to test the application's behavior in a realistic environment and chose the Nokia series S40, S60 and S80 in this context. To access the movie catalog and the movie itself, we used the HTTP protocol on the client and server side. It supports useful features like resuming downloads. The great advantage of this choice is the full support through MIDP on the client side and the easy implementation on the server side using an Apache webserver.

### B. Components and Functionality

The components have been implemented in specific Java classes, as presented in Figure 2. The prototype video player contains the GUI structure for selecting the offered services,

downloading the catalog information, and for playing a selected movie. The StreamLoadBalancer class holds all the functional aspects and requirements to handle the streaming of the movie during run-time as shown in Figure 4.

### C. Simulation and Evaluation

The experimentation and simulation studies are set up as follows: Several pre-selected video streams with different sizes (which are big enough not to be stored on the mobile phone in one piece) are offered by the video service to be transmitted to the mobile client. The QoS requirements for the mobile client are detected before request. In our experiments we infer about the video player and connecting status for simulating forced and unforced interruptions. The other QoS parameters are not modified.

We present the following experiments in which the user requests the best possible service quality:

- 1) *Static run*: The video player has been tested for different device settings with regard to the best possible QoS parameters, such as best connection, highest CPU usage, etc. The file size was varied from 10MB to 100 MB. The experiments have shown robust and good performance for handling large movie streams.
- 2) *Interruption run*: The same setting as before except that the connection was interrupted by the user during live streaming and restarted at the same position or the connection to the server broke (simulated). Thus, the client stopped playing until a reconnection occurred and the video stream continued playing at the end of the broken connection. The experiments showed that, according to the device characteristics (especially local storage), the user will not experience any unforced disconnection error for a reasonable amount of time.

## V. RELATED WORK

Video streaming on mobile devices using J2ME is a huge challenge for developers. Those devices do not support platform neutral video streaming, because they depend on native architectural requirements. We addressed our work to present a solution, which works for all Java-enabled mobile devices. Some manufacturers, like Nokia, deliver their products with

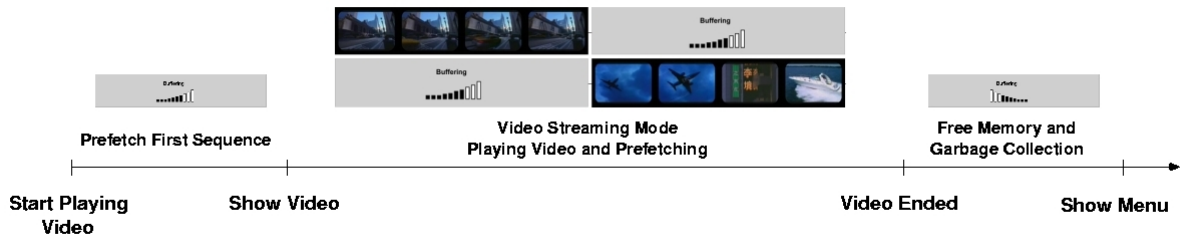


Fig. 4. Prefetching of Video Player during runtime

non Java-based players with streaming ability such as Real Player by Real Networks [12].

Prefetching video data was discussed by Fitzek and Rei [13]. They proposed a protocol to handle the pre-fetching mechanism depending on the error rate of the wireless links. We do not consider transmission errors and focus on pre-fetching for different device settings in case of interruptions. In [2], pre-fetching is seen as a suboptimal alternative for video streaming. Liu and Nelakuditi assume that a service would have to hold the necessary memory permanently. In our setting, we have to fill the stream buffer if the first part of the video is displayed, our adaptive service detects the free memory size, and changes the size of the video data packages and buffer allocation.

Several research studies on video streaming closely relate to our work. [1] and [14] presented adaptive services for QoS settings and context awareness. They focused on the device specific encoding of video streams. Architectural frameworks were proposed in [15] and [16]. The key requirements for mobile services in heterogeneous environments were identified. Our work can be considered as a realization of these papers for platform-independent video streaming. Using a CORBA middleware approach as in [17], the control of QoS in multimedia application environments is addressed.

## VI. CONCLUDING REMARKS

We presented a platform-independent adaptive approach for mobile devices to process video streams with regard to various QoS requirements. We showed that our approach fulfils the set objectives, which were to create an adaptive service on the client side and provide platform independence. The latter allows us to port our results on any newly introduced Java-enabled mobile device in the future and to improve our prototype successively.

First, we defined the research environment and introduced our prototype's architecture on the client side. After that, we developed a video streaming prototype derived from this architecture, which shows the practical relevance and utility of our theoretical model. Last, we set our work in relation to former research.

Currently, we are working on several extensions to the video streaming prototype, like increased QoS support for all QoS parameters or more adaptive disconnection policies. In

future work we will study our experimental work in real-life scenarios and environments.

## REFERENCES

- [1] O. Davidyuk, J. Riekkii, V.-M. Rautio, and J. Sun, "Context-aware middleware for mobile multimedia applications," in *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*. ACM Press, 2004, pp. 213–220.
- [2] T. Liu and S. Nelakuditi, "Disruption-tolerant content-aware video streaming," in *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*. ACM Press, 2004, pp. 420–423.
- [3] HTTP, "Hypertext transfer protocol: <http://www.w3.org/protocols/rfc2616/rfc2616-sec1.html>," 2005.
- [4] XML, "Extensible markup language: <http://www.w3.org/xml/>," 2005.
- [5] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services. Concepts, Architectures and Applications*. Springer-Verlag, 2004.
- [6] C.-H. W. Lund and M. S. Norum, "A framework for mobile collaborative applications on mobile phones," Norwegian University of Science and Technology, Tech. Rep., 2004.
- [7] J2ME, "Java 2 micro edition: <http://java.sun.com/j2me/>," 2005.
- [8] C. Marchetti, B. Pernici, and P. Plebani, "A quality model for multichannel adaptive information," in *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM Press, 2004, pp. 48–54.
- [9] R. Braumandl, A. Kemper, and D. Kossmann, "Quality of service in an information economy," *ACM Trans. Inter. Tech.*, vol. 3, no. 4, pp. 291–333, 2003.
- [10] M. Wermelinger, G. Koutsoukos, H. Lourenco, R. Avillez, J. Gouveia, L. Andrade, and J. L. Fiadeiro, "Enhancing dependability through flexible adaptation to changing requirements," *Lecture Notes in Computer Science*, vol. 3069, p. 3, 2004.
- [11] P. D. Ezhilchelvan and S. K. Shrivastava, "A model and a design approach to building qos adaptive systems," *Lecture Notes in Computer Science*, vol. 3069, pp. 215 – 238, 2004.
- [12] R. Networks, "Realplayer: <http://www.real.com/>," 2005.
- [13] F. Fitzek and M. Reisslein, "A prefetching protocol for continuous media streaming in wireless environments," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 10, pp. 2015–2028, 2001.
- [14] J. R. Taal, K. Langendoen, A. van der Schaaf, H. W. van Dijk, and R. I. L. Lagendijk, "Adaptive end-to-end optimization of mobile video streaming using qos negotiation," in *Proceedings of international symposium on circuits and systems (ISCAS 2002); special session on Multimedia over Wireless Networks*, 2002.
- [15] P. Inverardi, F. Mancinelli, and M. Nesi, "A declarative framework for adaptable applications in heterogeneous environments," in *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*. ACM Press, 2004, pp. 1177–1183.
- [16] B. Althun and M. Zimmermann, "Multimedia streaming services: specification, implementation, and retrieval," in *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*. ACM Press, 2003, pp. 247–254.
- [17] R. E. Schantz, D. C. Schmidt, J. P. Loyall, and C. Rodrigues, "Controlling quality-of-service in a distributed real-time and embedded multimedia application via adaptive middleware," <http://www.cs.wustl.edu/schmidt/corba-research-realtime.html>.