STREAMED MULTIMEDIA PRESENTATION FOR LOW-BANDWIDTH MOBILE TERMINALS: A VIRTUAL MACHINE APPROACH

Lars Aarhus, Håvard Hegna, Thor Kristoffersen,
Wolfgang Leister, Anders Moen, Bjarte M. Østvold
Norwegian Computing Center
Postboks 114 Blindern,
NO-0314 Oslo, Norway.
`http://www.nr.no`

Abstract

We present a simple and robust client–server architecture for streaming synchronized multimedia presentations to mobile terminals over a low-bandwidth connection.

The server is similar to a compiler, and translates high level SMIL descriptions of multimedia presentations into low level content instructions. The client resembles a virtual machine, operating on content instructions that control the multimedia presentation. The compilation process on the server is designed to reduce the buffering on the client, and hence reduce the start-up delay for the user. Since the server takes the larger burden of the total work, the client can be kept simple.

Upon request, the server assembles a sequence of content instructions corresponding to a multimedia presentation, under the optimistic assumption of a minimum communication bandwidth, and streams the content instruction sequence to the client without any other feedback mechanism than flow control. If the bandwidth assumption holds, the client renders the presentation as specified; otherwise, it is able to make controlled pauses in the presentation.

We also describe a specific news application based on our architecture.

1 Introduction

Until recently streamed multimedia applications have not been widely available for low-bandwidth mobile terminals. Especially video bandwidth requirements have not been supported by current mobile telecom networks such as Global System for Mobile communication (GSM). With the introduction of higher bandwidth and packet based mobile networks such as Universal Mobile Telecommunications System (UMTS) this is expected to change. However, next generation mobile telecom networks will also have varying bandwidth depending on location. Mobile terminals will have limited resources compared to desktop terminals on wired networks. Streamed multimedia presentation in mobile networks will still be a challenge.

In this paper we present a client–server based system architecture supporting *stretchable* streaming of synchronized multimedia presentations on mobile terminals. By stretchable we mean that controlled pauses may be introduced by the client at suitable times in the presentation, in order to allow smooth continuation of the presentation under low-bandwidth and varying network conditions. Our architecture does not use an application level feedback mechanism for stream control, but the stretchability compensates for this. A main design goal for the architecture is reducing the start-up delay for the user.

The paper is organized as follows: The current section motivates our architecture by an example application. Section 2 describes related architectures. In Section 3 our approach to the design challenges is presented, which in Section 4 leads to a description of the overall system architecture on a conceptual level. Sections 5 to 7 conclude our work by discussing the architecture and describing the prototype implementation.

1.1 The News Reader Application

An example news reader application has been developed to demonstrate the proposed architecture. The application idea is for a user on a hand-held terminal to get up-to-date multimedia news items on demand. The news items are supplied from a server over a low-bandwidth network. Each transferred item consists of both continuous (audio, animation) and static media (images, text), which are presented synchronously on the mobile client. A 3D animated head, accompanied by voice audio, is used to simulate a TV news anchor's face reading the news.

We use data compression for all data types. The use of an animated news reader also constitutes a form of compression: animation information take up very little bandwidth compared to video.

The application user interface is simple, and allows the user to select a news item from a menu list of available items offered by different news channel. Available items are dynamically updated on the server. An on-going presentation can be stopped upon request by the user. Each mobile client connects to the server individually, on a one session

per client basis.

Up-to-date content is stored in a database accessible to the server. The content is built around audio clips of human voice. Corresponding lip movements for the animated head are extracted, while suitable head movements and eye blinking are generated automatically. Images and a textual summary of the spoken words are then added to the presentation.

News items are represented using the Synchronized Multimedia Integration Language (SMIL) [10], which is an XML document type definition. A SMIL specification structures a multimedia presentation into sequential and parallel schedules that can be nested to any depth. Relative timing and spatial information can be added. The basic building blocks are media elements that refer to the media files.

2 Related Architectures and Technologies

Common approaches to streamed multimedia in varying, low-bandwidth network environments use layered coding techniques for continuous and static media. The layered *continuous* media approach is used in both RealSystem iQ streaming, and MPEG-4 coding described below. This could include adapting the media data rate to the network characteristics, which requires dynamically monitoring network parameters such as delay and jitter to avoid buffer underflow [8].

Turner and Ross [9] find the optimal number of layers for *static* media in order to maximize the overall presentation in bandwidth-limited networks. They assume that the bandwidth of the continuous media is constant, and the presentation time-line is fixed.

Georganas et al. present a multimedia news on demand application [1, 4]. The system relies on feedback over the network. It operates with a fixed time axis on the client, and must therefore delay or drop frames in case of starvation threats from the network.

On *wired* Internet there are many streaming media architectures. Two dominant commercially available technologies are RealSystem iQ[1] from RealNetworks, and Windows Media Technologies[2] from Microsoft. Both are client–server architectures with point-to-point connections that use proprietary streaming media formats, with typical data rates downwards limited to 28.8 kb/s or 56.6 kb/s.

RealSystem iQ is of particular interest as it supports SMIL content presentation streaming, as well as Real Time Streaming Protocol (RTSP) for client-driven application-level media server control.

The MPEG standards are relevant to our work. The MPEG-2 [2] transport stream (MPEG-TS) is designed for feedback-less broadcast and involves multiplexing of various multimedia streams into a single transport stream. These features also inspired our architecture. MPEG-2 is intended for digital TV with orders of magnitude higher bandwidth requirements.

MPEG-4 [3] is used for distributing general layered multimedia objects, and is designed for lower bandwidth links down to video bit rates of 10 kb/s. MPEG-4 has characteristics that makes it suitable for use when distributing multimedia to mobile terminals [6]. However, both MPEG architectures try to cover many areas of usage so that they cannot efficiently address typical requirements for use on client machines with scarce resources.

3 Design Choices

In this section we present the challenges posed by the news reader application. We then address the design choices that we made for our architecture in order to meet those challenges.

3.1 Challenges

The nature of the news reader application, and the physical constraints imposed by typical mobile terminal hardware and wireless networks, pose serious design challenges [7].

1. The client should start multimedia presentations as soon as possible and presentations should not stop at *random times*. In the case of the news reader application this gives users a “news on TV” feel.
2. Client hardware resources are limited, especially processor speed and memory. Consequently client-side processing must be simple.
3. Wireless networks have limited bandwidth, and in general their quality varies over time in unpredictable ways. They suffer from greater risk of packet loss than wired networks, mostly due to physical conditions, e.g., rain, and user motion [11].

Of these challenges number 1 represents the requirements of our news reader application, while challenges 2 and 3 arise from physical constraints.

3.2 Approach

Our system architecture is based on the following design choices. The choices address the indicated challenges from the previous section:

Challenge 1. In this class of applications, solution strategies are divided into two classes: *preventive*, i.e., strategies that attempt to avoid problems, and *corrective*, i.e., strategies that attempt to solve problems when they occur [4]. To address challenge 1 we employ both types of strategies.

With respect to the synchronization reference model, multimedia content transfer is streamed at the stream layer and at the object layer. This enables the client to start a multimedia presentation even if it has not yet received all the timestamped content data that the presentation consists of. This scheme requires, however, that the client knows when

[1] `http://www.realnetworks.com/`

[2] `http://www.microsoft.com/windows/windowsmedia/`

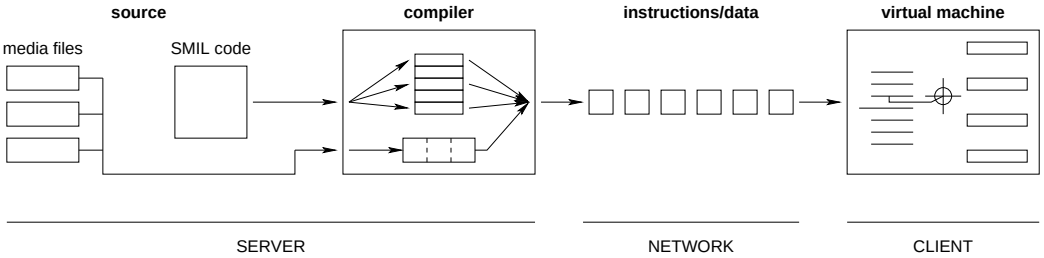

Figure 1 System architecture.

it can start the presentation without the risk of running out of data. We provide such a mechanism following a preventive strategy based on server-side calculations which assume knowledge of the communication bandwidth within a safety margin.

In our application, audio data have the largest volume among multimedia data, and therefore we treat it specially. The audio is split into segments at points where there is a natural pause in the news reader's voice, e.g., at the end of a phrase. The result is a series of non-stretchable audio segments punctuated by stretchable pauses. This enables the client to employ a corrective strategy by enlarging these pauses if it is starved of data on the network connection.

Challenge 2. The server translates SMIL multimedia presentations into smaller, low-level units of content with timestamps before sending it to the client. Therefore the client-side of the application can follow simpler rules to analyze and display multimedia content, and have simpler data structures (no SMIL representation).

Challenge 3. For data transport we use a single Transmission Control Protocol (TCP) connection per client, where different media types are compressed and multiplexed into one socket stream. This choice is a pragmatic one, since the inherent reliability of the protocol makes the architecture implementation simpler. Also, the use of TCP means that the server cannot overrun the client, and the link is kept saturated from the start.

4 Multimedia Compiler and Virtual Machine

This section describes the overall system architecture, which is illustrated in Figure 1. The figure shows the analogy between the following two sets of entities: (i) The server and its multimedia content, the network and a client playing the content, and (ii) A compiler and its source files, instructions (code) and data, and a virtual machine executing the instructions.

The server transmits to the client a sequence of multimedia content instructions, that make up an entire multimedia presentation corresponding to a news item. These instructions constitute a compiled version of the SMIL sources and media files, tailored for interpretation on the virtual machine. The virtual machine interprets low-level content instructions that convey the multimedia data to be rendered, as well as precise information about how, where, and when to render it.

4.1 SMIL Compiler

While the client is largely insensitive to the order of the content instructions that it receives, the exact sequencing is important with respect to reducing the waiting time for the user before the presentation starts. Since most individual content instructions are not sensitive to the order in which they are executed, careful scheduling of items can be used to achieve streaming at the presentation level. As an extreme example of suboptimal scheduling, if the opening image of the presentation is transmitted last in the content, the user will have to wait until the entire content has been transferred before the presentation can start.

The news content is stored persistently on the server in a hierarchically structured multimedia database. Before transfer to a client, a news item is compiled into a content sequence of virtual machine instructions suitable for network transmission. This compilation consists of several passes.

The first pass, *flattening*, generates timestamps for the instructions. In a SMIL schedule, all timings are relative to one another, so that for instance it is possible to swap two elements in a sequential schedule without the need to specify a different timing. The virtual machine, however, expects the instructions belonging to one presentation to be timestamped relative to the start of the presentation. The result of this pass is a sequence of timestamped multimedia items, i.e., the SMIL structure is removed.

The second pass, *splitting*, splits the multimedia items into content instructions. There are four instruction types: `zero`, `atomic`, `render`, and `fragment`.

The `atomic` instruction is for small volume multimedia data like text, whereas the `render` and `fragment` instructions are used together for large transfers, like streaming audio. In large volume transfers multiplexing and temporal optimization is made possible by splitting data into `fragment` instructions: one `render` instruction is followed by a sequence of `fragment` instructions.

The third pass, *reordering*, reorders the instructions so that their order is optimal for the virtual machine, i.e., they

should be sent in the sequence that minimizes the start-up delay for the user. An important factor in deciding the order of the instructions is the timestamp. The algorithm employed here is similar, but not identical, to the algorithms described by Paul et al. [5].

The fourth and final pass, *zero placement*, places the `zero` instruction within the constructed sequence, so that the presentation can start as soon as it is reasonable to assume that the virtual machine will not run out of data.

The resulting sequence of instructions is encoded in a low-level client–server application protocol format and transmitted to the virtual machine. Since the client cannot provide any timely feedback to the server at the application level, the channel bandwidth is assumed to be constant within a safety margin, and the compiler bases its computations on this model bandwidth. A consequence of this is that the instruction sequence does not need to be computed more than once for each unique multimedia presentation.

4.2 Content Instructions and Media Data

The multimedia content is transferred to the client in content instructions and file fragments. File fragments with image and audio content are assembled to files, and decompressed upon their arrival on the client. Content instructions may refer to these files, e.g., `render` instructions referring to images to be rendered.

The instruction format contains time-, location- and opcode information, including parameters for the instruction:

seqno:timestamp	region	opcode	params

The multimedia content is organized in numbered content sequences, each having a time axis from 0 to infinity. A sequence is content-dependent, and usually corresponds to one news item. The instructions of a content sequence have a timestamp that is relative to the start of this sequence.

The opcode and the parameters of an instruction give the semantics. The display of the terminal is divided into non-overlapping regions. For media-related instructions the media type is encoded in the opcode.

4.3 Virtual Machine for Playing Multimedia

The client is a virtual machine (VM) capable of executing content instructions and using media data and thereby executing, or playing, a multimedia presentation (see Figure 2). The virtual machine translates content instructions to micro instructions, and controls scheduling and processing of micro instructions. The formats for content and micro instructions are similar. The instructions are grouped into different types (see Table 1). Micro instructions have their opcode and parameters organized such that no further parsing of the content is necessary.

The play unit controller is responsible for scheduling instructions and data, and addressing the proper processing units. The processing units are called pipes, one for each region and opcode. Pipes render data, e.g., a raster image, text on top of a raster image, or play audio. Animations, e.g., facial animations, are driven by special pipes, called periodically by the operating system, that feed animation instructions to the animation logic.

Type	*Examples*	*Semantics*
`atomic`	`MSG, TXT`	content instructions for messages, and formatted text.
`state`	`ACT, SUS`	instructions for control of operation.
`control`	`JMP, JIN`	continue playing at new content sequence and timestamp.
`resource`	`EXC, RCV`	handle instructions connected to resources.
`render`	`RND, VIS`	render a media object (e.g., images (`RND RAS`), audio (`RND AUD`), and different types of visemes).

Table 1 VM micro instruction types.

All incoming instructions are stored in the instruction list, which is sorted in temporal order. The instructions in the instruction list are organized in content sequences, and grouped into play units. Play units are pieces of a sequence which can be played without experiencing lack of resources (e.g., missing images). Instructions are processed when the processing-time has arrived according to the timestamp. However, some instructions are processed immediately upon arrival, in order to achieve an external interrupt (e.g., `JMP` instructions).

The virtual machine operates in cycles that are called regularly by a timer. In each cycle it checks whether the first instruction in the instruction list is to be processed. The pipes execute the instructions.

4.4 Virtual Machine Operation

During operation, the VM is in one of two states, called *active* and *suspended*. These states exist also as instructions, in order to achieve a transition into a new state.

The active state indicates that the machine plays content. An active-instruction indicates the start of a new play unit, which determines when to play its content. The play units are generated internally in the client using a heuristic.[3]

All content preceding the current active-instruction is always deleted, while all content between the timestamp of the current active-instruction and the current time is played immediately.

The suspended state indicates that the VM currently waits for a resource to be satisfied, e.g., media content to arrive. Instructions before the current suspend-instruction are removed from the instruction list, while instructions after the suspend-instruction are not processed. In suspended state no

[3] In our implementation we found it useful to let the splits in the audio content guide the start of a new play unit.

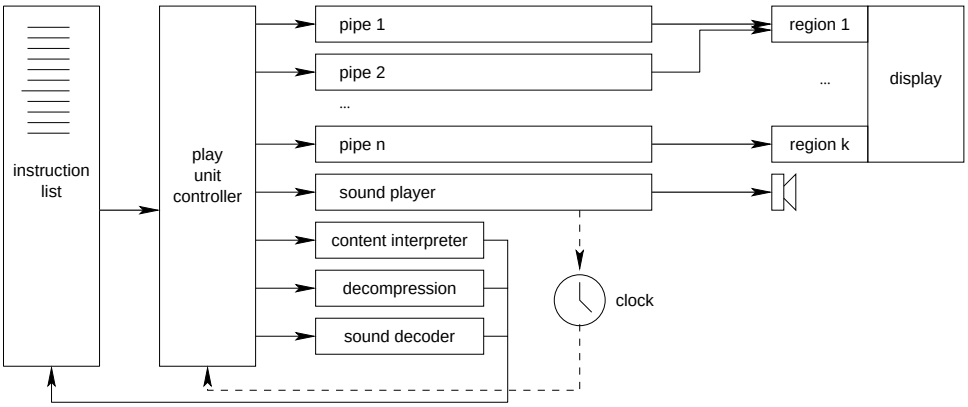

Figure 2 Virtual machine.

new instructions are fed to the pipes.

When a resource is needed, that has not been satisfied, a suspend-instruction is generated, while the arrival of such a resource generates an active-instruction.

4.5 Code Generation in the Virtual Machine

Content instructions are translated into micro instructions upon arrival by the content interpreter. For an example of the translation process, we refer to Figure 3. The VM has the following mechanisms to generate instructions:

- The arrival of media files and content instructions (especially instructions to display content) trigger generation of `state` instructions.
- The payload of content instructions is parsed and translated into micro instructions.
- The client is capable of playing pre-stored content blocks (jingles). This content is copied into the instruction list upon request.
- Visemes are animation instructions for the head of the news reader. These are transferred as file fragments from the server, and compiled on-the-fly to a great number of animation micro instructions.

5 Discussion

Our architecture is similar in function to that of a SMIL player, but our player is distributed, consisting of a server and a client part. This allows the client to be light-weight, but assumes the existence of a cooperating server on a separate machine.

The architecture is suited for applications without absolute real-time characteristics, e.g., applications where we can afford to wait for the arrival of content for a certain, reasonable time. There is no media quality based degradation policy.

In our architecture the client can stretch the presentation in time. In this way we handle varying network characteristics. Consequently, the duration of multimedia presentations played by the client is no longer fixed.

There is no feedback from the media rendering devices back to the scheduling logic, because the round-trip time over a mobile network link is relatively large. The client can show the user special pause content while waiting, thus avoiding the sudden freezing of the presentation.

Of the related approaches and architectures previously discussed, that of Turner and Ross [9] is most similar to ours since both approaches aim at low-bandwidth applications. The main difference is that their approach has a fixed time axis which does not support stretchable streaming. They also use layered media coding techniques.

Georganas et al.'s news-on-demand system [1, 4] operates on MPEG video media objects in addition to other media since it is tailored towards broadband, not low-bandwidth networks. Their architecture assumes the existence of feedback mechanisms for stream control.

The other architectures are mainly available for *wired* or in some cases wireless LAN, not low-bandwidth mobile telecom networks.

One advantage of our architecture is the small footprint it occupies on a mobile terminal. This is in contrast to encompassing architectures like MPEG or JMF which are not sufficiently tailored for, or efficient enough for, those kinds of environments.

6 The Prototype Implementation

The Ericsson *Communicator* platform was used to implement our concept. This platform supports GSM communication at 9.6 kb/s, and is equipped with a 190 MHz StrongARM processor and 16 MB RAM. The system software is the EPOC operating system, which is designed specifically for small footprint hardware.

The server based its optimistic planning of instructions and data transfer to the virtual machine on having a steady bandwidth of 7 kb/s. This was sufficient for controlling the client and less than the available bandwidth. Audio streaming was based on 5.6 kb/s half-rate GSM audio coding, thus

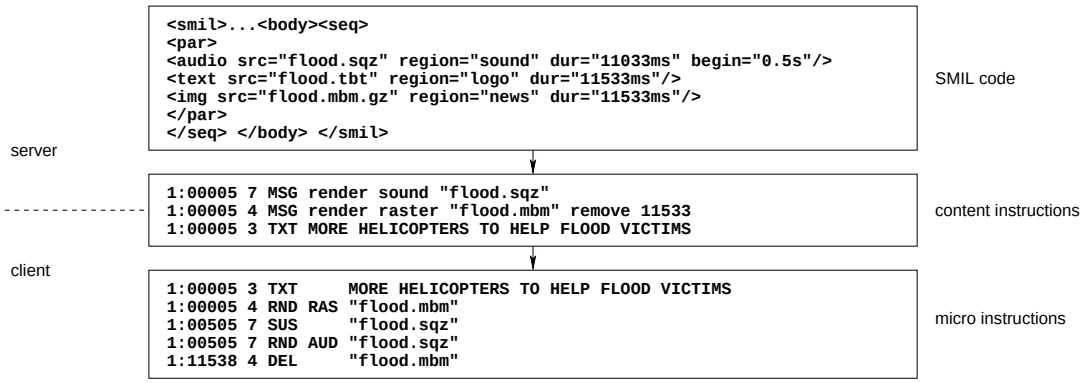

Figure 3 Code translation.

2.6–3.0 kb/s was left for the other media.

The news reader application was demonstrated at the CeBIT 2000 business fair. At regular intervals a production team produced news content in the required format, and uploaded it to the server. News content was supplied by wire services.

7 Concluding Remarks

Our system architecture represents a general approach to the problem of delivering multimedia presentations to mobile terminals on a low-bandwidth connection. Scheduling SMIL content on the server unburdens the client of this task. This means that the client implementation can be sufficiently lightweight for a mobile terminal with scarce resources.

The architecture implements streaming, and incorporates both preventive and corrective strategies for this. As a preventive measure, the server optimistically schedules the content instructions such that everything looks fine if the bandwidth assumption holds. If the bandwidth assumption does not hold, the client virtual machine makes a controlled pause in the presentation.

Acknowledgements

Besides the authors, Dag Belsnes, Arne-Kristian Groven, Arve Larsen, Shahrzade Mazaher, Jonn Skretting, and Håkon Steinbakk participated in the design and implementation of the news reader application. Fredrik Crawfurd was our unfailing hardware guru, and Steinar Kristoffersen lead the project. Jason Baragry, Peter Holmes, Arve Larsen and Olaf Owe provided valuable feedback on earlier versions of this paper. The project was financed by Mobile Media.

References

1. M. Daami and N. Georganas. Client based synchronization control of coded data streams. In *Proc. Intl. Conf. on Multimedia Computing and Systems, (ICMCS'97)*, pages 387–394. IEEE, 1997.
2. International Organization for Standardization. *MPEG-2 – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s*, 1993. ISO/IEC 11172-1:5.
3. International Organization for Standardization. *MPEG-4 – Coding of audio-visual objects*, 1999. ISO/IEC 14496-1:6.
4. J. Jarmasz and N. Georganas. Designing a distributed multimedia synchronization scheduler. In *Proc. Intl. Conf. on Multimedia Computing and Systems, (ICMCS'97)*, pages 451–457. IEEE, 1997.
5. R. Paul, M. F. Khan, and S. Baqai. Real-time scheduling for synchronized presentation of multimedia information in distributed multimedia systems. In *Proceedings of the 3rd Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97)*, pages 177–184. IEEE, 1997.
6. A. Puri and A. Eleftheradis. MPEG-4: An object-based multimedia coding standard supporting mobile applications. *Mobile Networks and Applications*, 3:5–32, 1998.
7. M. Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 1–7. ACM, May 1996.
8. Y. Song, M. Mielke, and A. Zhang. Netmedia: Synchronized streaming of multimedia presentations in distributed environments. In *Proceedings of IEEE Multimedia*, 1999.
9. D. A. Turner and K. W. Ross. Optimal streaming of synchronized multimedia presentations with layered objects. In *Proceedings of IEEE International Conference on Multimedia and Expo*, 2000.
10. World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*. REC-smil-19980615.
11. G. Xylomenos and G. C. Polyzos. Internet protocol performance over networks with wireless links. *IEEE Network*, 13(4):55–63, 1999.