INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO INFORMATION

ISO/IEC JTC1/SC29/WG11/**M2286**
MPEG97
Stockholm/July 1997

Source:	**Ecole Nationale Supérieure des Télécommunications (ENST)**
Status:	**for discussion**
Title:	**Synchronization simulator for concurrent MPEG-4 terminals**
Author:	**Samuel Tardieu, Laurent Pautet, Jean-Claude Dufourd**

Introduction

Goals

Synchronization of different input sources is a trivial problem when the decoding time is zero, the CPU is infinite and the resources are unlimited. Unfortunately, this model is not even close from what real decoders provide; they have a limited CPU, limited resources, and the decoding time of an object may not be predictable with a fine grain.

Our demonstration tries to verify the validity of the systems decoder model while considering all the above-mentioned constraints. In particular, we are studying the various strategies:

- what is the benefit of having some meta-information about the predicted decoding time in the bitstream?
- is it possible to build a coherent system with “ intelligent ” decoders that are able to detect a possible lack of resource (including CPU time) and that can take actions to run in degraded mode without disturbing the whole system too much?
- if the intelligence is centralized in the compositor, what kind of algorithm is best suited for this work? (EDF == Earliest Deadline First, LLF == Lowest Laxity First)
- how can we implement graceful degradation if CPU and resources are lacking? Can we keep the same strategy for every kind of object?

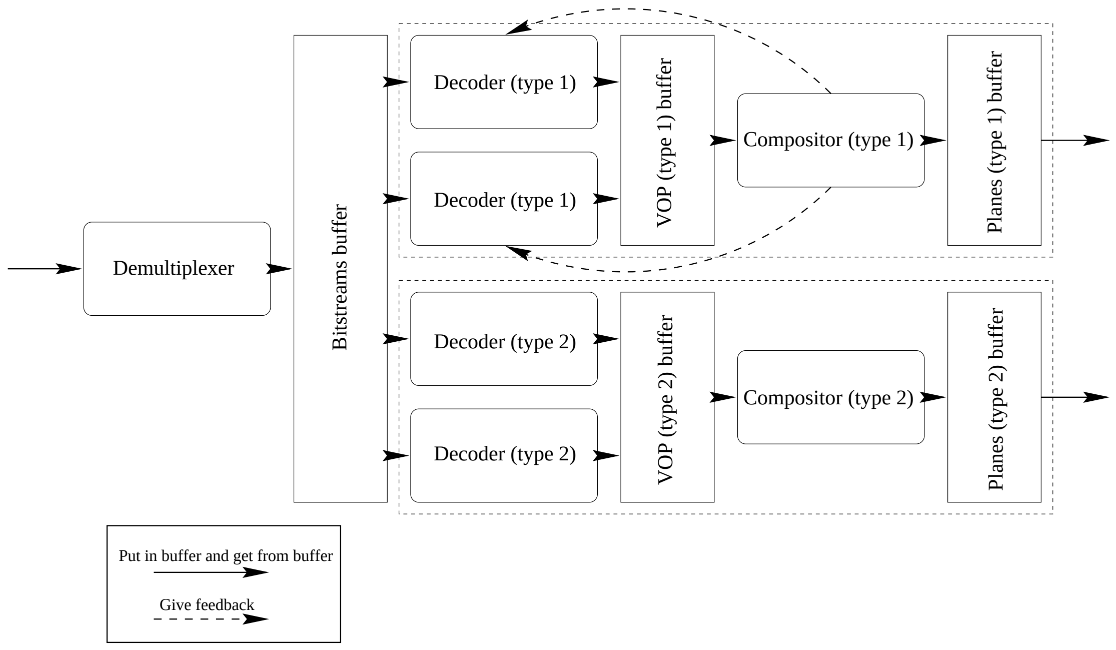
Framework and development environment

The system used was a Toshiba laptop (P133) running Linux RedHat 4.1 and X11R6.3. We chose to use the Ada95 programming language because of its native support for tasking and its high-level synchronization paradigms (Hoare monitors, rendez-vous, etc.); the compiler used was GNAT 3.10, a free Ada compiler based on GCC, the well-known compilation system (available for many targets, as a native- or as a cross-compiler).

In order to give a feeling of the results of a particular strategy or combination of parameters, we have included a video output for the simulator. It presents a series of pre-decoded and pre-loaded frames for two or four video objects (coming from one video sequence split into two or four and to which cursors have been added to make it easier to see mis-synchronisation, if any). It mirrors exactly how the simulated terminal works.

Simulator architecture

The next figure shows how our prototype is implemented:



Each logical block is made of a separate Ada task which communicates with other blocks through synchronized buffers (also called Hoare monitors). Each time a buffer is in an underflow or overflow state, the demonstrator records it for analysis purposes.

> Warning: in the following, the word *decoder* is used for a *hardware decoder*, or a processor. Telling the simulator to use one decoder means that one processor does all the decoding, whether it runs a video decoding software or audio decoding software or SNHC decoding software. When more processors are used, only the decoding load is spread among the processors (which have all the same capabilities).

Let us have a walk-through of the workings of the simulator. The input is a series of object plane descriptions, containing:

- object ID
- plane number
- decoding time for this particular plane (this information is used for the simulation, but not for the scheduling)
- type of object (high/low priority, video/audio type of degradation…)

The object planes are sequential within an object, but are not ordered between different objects. The decoding times vary randomly between a minimum and a maximum value. These input files are generated by a separate program and stored into a file to have the same input for various runs of the simulator. For the moment, the realism of the input files is not very good: we have only used quite extreme cases.

The demultiplexer reads in the input sequence and dispatches the pseudo access units to the decoding buffers. The decoders read in the information, sleep for the said decoding time, and wake up to put the “decoded” plane in the composition buffer. As processing time (the amount of time for the decoder task to read in information and decide how long to sleep) is very small, the number of processors that can be simulated is quite high. The compositor monitors the content of the composition buffer and implements the synchronization / degradation strategy.

The type of object is used to determine how to proceed with degradation. Our current strategy protects the audio objects and penalizes the video objects if necessary. Planes from objects within the same level of priority will be synchronized, but older low-priority object planes may be presented together with newer high-priority object planes.

The number of different object types can vary. The number of decoders can be changed as well via command line arguments. Given that decoders are mapped onto Ada tasks and tasks are mapped onto processors, increasing the number of processors will automatically decrease the load of each of them: the various tasks will be dispatched automatically on more processing units. There is no need to recompile the application or even to reconfigure it. It will automatically use all the CPUs it has been asked to use.

In this model, one processor will be enough to decode three different kinds of input streams (video + audio + snhc data) since the system will schedule the various decoders given a fixed algorithm. If there is not enough computation power, then the system will enter a degraded mode, according to the indication that will be found in the bitstream or in the terminal strategy (for example “ preserve the audio as much as possible ”). Another decoder with three processors will have no problem to decode everything smoothly with the very same software, which shows an immediate benefit.

Preliminary results

Note: the tests we did until now do not represent real cases. The test suite goals were to make sure that the simulator was running properly and was putting out meaningful and usable values.

We built different input streams with different types of VOPs whose decoding times could vary up to two orders of magnitude. We have noticed that modifications in the buffer sizes (typically increasing the intermediate buffers, to allow for an initial latency to fill up some buffers) could dramatically increase the overall performances when decoding times are varying a lot in a single bitstream. On the other hand, when decoding times are similar, small buffers are sufficient because the variations of the “ fullness ” of each buffer are quite small.

We also tried different degradation strategies, and found, as a preliminary result, that two different ones were very useful and comfortable for eyes and ears. Both share the same basis. If you have a buffer containing ‘N’ planes (for example ‘N’ pictures ready to be displayed), and the compositor for this type of objects notices that there is less than ‘a’ planes in advance (for example we are displaying the picture 123 and pictures up to 126 are in the buffer, and ‘a’ is 5, so we consider that at this time we should have 128 ready), then the compositor asks the decoders not to decode anything more before the frame ‘R+a+k’, where ‘R’ is the latest frame ready and ‘k’ the skip factor.

For video, it seems to be best to keep ‘k’ quite small (for example 2) not to skip too many frames at a time. It seems better to get one frame out of three than 10 successive frames each 30 frames.

For audio, ‘k’ must be greater, because it is best to skip half a second of audio and go on with a continuous stream instead of playing one sample out of three.

Conclusion and future work

Our source code as well as Linux and Solaris executables will be made available on the Emphasis FTP site shortly. The simulator will be improved to allow a finer instrumentation. The input generator will be fine-tuned to generate more meaningful input sequences. We will also try to gather real-life numbers for our input sequences. We will study in particular:

- the impact of decoding time variability on the latency need,
- the impact of object plane ordering on buffering needs, and the associated requirements on the encoder,
- the efficiency of parallelization…

More results should be submitted to the next MPEG-4 conference.