On existing Unix SVR4 systems, the model we described could be integrated as a new Multimedia Scheduling Class, mapped into the global priority vector of the system. The integration of the model into a split-level like [2] structure is also an interesting topic.

4 References

[1] Bourges Waldegg, D., Lagha, N. and Le Narzul, J.P.- Multimedia Applications on a SVR4 Kernel: Performance Study. *In Proceedings of the 3rd COST 237 Workshop on Multimedia Telecommunications and Applications*.- Barcelona, november 1996

[2] Govindan, R., and Anderson, D.- Scheduling and IPC Mechanisms for Continuous Media. *In: Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, SIGOPS, Vol. 25, pp 68-80.- Pacific Grove, California, USA, 1991

[3] Little, T.D.C., Ghafoor, A., Chen, C.Y.R. and Berra, P.B. – Multimedia Synchronization. *IEEE Data Engineering Bulletin*, Vol. 14, nº 3, pp. 26–35, Septembre 1991

[4] Sénac, P., Diaz, M., de Saqui-Sannes, P.- Toward a Formal Specification of Multimedia Synchronization Scenarios. *Annals of telecommunications*, Vol. 49 nº 5-6, 1994

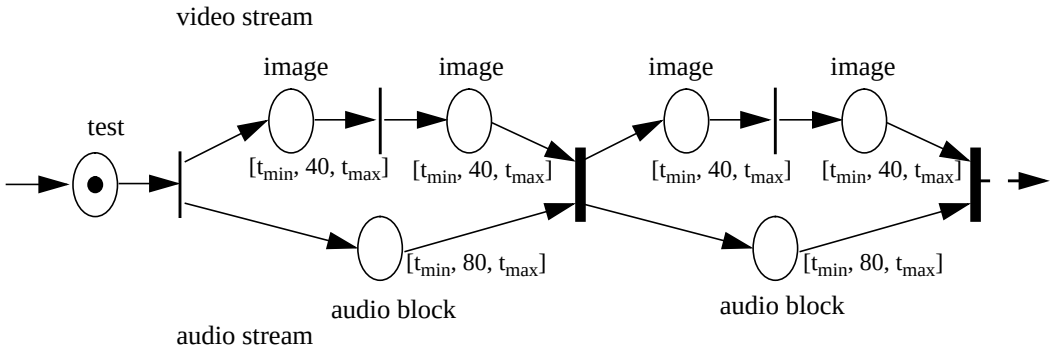

FIGURE 1. a TSPN exemple

The marking of a place is associated with the first scheduling of the thread that treats the information unit, and a transition firing means that presentation data is available (thus a presentation thread needs to be scheduled); if a presentation thread is not ready to run when the transition is fireable, then the corresponding scaling thread is awakened and scheduled.

An absolute priority is associated to each stream according to the synchronization type defined at synchronization points. For instance, a stream acting as a synchronization master will have a higher priority than slave streams.

At each scheduling instant, a relative priority is assigned to the thread (reading, processing or presentation) corresponding to the stream whose dynamic temporal validity interval is closer. A time quantum is also assigned to each stream, so that a regular distribution of occupation time is achieved between synchronization points (to prevent from degradations due to long inter-scheduling periods). The choice of the thread to be scheduled depends then on both absolute and relative priorities.

A transition, meaning that the unit can be presented, is fired when the scheduling instant belongs to the transition's dynamic temporal validity interval. If processing has not been finished, the presentation thread will not be ready to run (as it has no data) and the scaling thread is awakened and scheduled in order to determine the actions to undertake (some actions could compute a new set of static temporal validity intervals).

Reading and processing threads can be scheduled in advance and the read/produced data be temporarily buffered; on the contrary, presentation threads need to be ready at specified instants

2.4 Scaling policy

The actions to be undertaken by the scaling thread are stream-dependent. Different kinds of scaling may be defined. For video streams, a common scaling mechanism consists of an image rate modification (frame rejection to allow for more decoding time). This mechanism is not based on a special coding an is relatively simple (it only has to consider frame dependency), but the result may be unsatisfactory for user perception. Other scaling techniques are inherent to data coding, like layered video in MPEG2 video coding.

3 Conclusions

The fact that the system scheduler is aware of synchronization requirements between data streams can help adaptive applications to take a better advantage of available CPU capacity. A model in which such a scheduling policy is applied has been sketched in this paper. On-going simulation of the model will help us define parameters such as intervals between synchronization points and adapted time quanta values for each media.

Current networks allow for the separate transport of individual streams so that channels adapted to media characteristics are used. As streams are presented simultaneously, a structure in which each individual stream is treated by a dedicated thread seems to be adapted.

Inside each stream, there are also separate activities, which do not necessarily take place simultaneously, but whose requirements are different. Therefore, it is useful to separate these activities in order for the scheduler to meet each one's need in the most appropriate way. We have defined the following generic types of threads:

- reading threads (from network/disk): can suffer from network jitter
- decoding/processing threads - may need high throughput, e.g., in the case of software video decoding
- presentation threads - in the case of continuous media, they need periodic attention; a QoS parameter can specify a tolerated drift between streams
- scaling threads - are used to apply the stream-dependent scaling policy

For each stream involved in the application, the programmer will have to define the activities to be performed by each type of thread. So the application will consist of one of these threads for each data stream. A combination of user level threads and kernel level threads can be defined in a split-level way, as proposed by [2].

2.2 Synchronization specification

Important work has been done in the field of multimedia synchronization specification. Petri Net Time extensions are particularly interesting since they can express presentation durations and synchronization points between streams in a simple yet powerful way.

Different kinds of synchronization relations can exist between streams (master/slave, and, or,...). These kinds of relations are summarized in [4]. For example, audio is less tolerant to intra-stream synchronization errors than video, hence an audio stream in an audio/video presentation can act as a master clock and therefore control the presentation of the video stream.

The Petri Net extension proposed by [4], called Time Stream Petri Net (TSPN) integrates synchronization types into transitions. As for processing or presentation durations, it associates a nominal duration to arcs outgoing from places, as well as a minimum and maximum duration times, defining thus an interval in which transitions can be fired. Specific firing rules are defined for each kind of transition. These time values, called *static temporal validity interval* for the firing of a given transition, allow the expression of variable processing times and network jitter.

For our scheduling policy purposes, we have associated places in the TSPN with the reading/processing/presentation threads of information units or blocks which compose individual streams. Synchronization points are defined at specific intervals, consisting of transitions with two or more input arcs. This interval needs to be defined according to a trade-off between synchronization error propagation and system overhead.

At a given instant *t*, a d*ynamic temporal validity interval* is computed, according to the transition type and the set of enabled transitions. A transition can be fired if it is enabled (i.e., all its upstream places are marked) and the instant *t* belongs to the computed dynamic validity interval.

An example of a TSPN specification of an audio/video sequence is shown in figure 1.

2.3 Scheduling policy

The aim of our scheduling policy is to respect non-scalable streams' deadlines, and to minimize degradation for scalable streams. Threads related to information units or parts of media which will not be presented will not be scheduled, since the scaling thread acts as a feedback loop controlling the reading thread.

The synchronization scenario specified by the programmer is notified to the scheduler (e.g., at application initialization), as well as the identifiers for the different threads associated to each stream. Scheduling decisions are based on this TSPN synchronization specification. During runtime, the scheduler updates the state of the TSPN each time a transition is fired, as well as a table containing all enabled transitions and their dynamic temporal validity intervals.

A Temporal Synchronization-based Scheduling Policy for Adaptive Multimedia Presentation Applications

Daniela Bourges Waldegg
Télécom Bretagne, Networks and Multimedia Services Department
BP 78, 35512 Cesson-Sévigné Cedex, France
bourges@rennes.enst-bretagne.fr

1 Introduction

In this paper we present the basic ideas for a multimedia presentation model in which the thread scheduling policy is based on an inter-stream synchronization scenario specified by the application programmer.

Temporal synchronization is a fundamental characteristic of multimedia data. Synchronization relations exist between different media streams (inter-stream synchronization) and between information units that compose a single stream (intra-stream synchronization) [3]. Inter-media synchronization is not considered by real-time scheduling policies, like Rate Monotonic or EDF, which are based on intra-stream time parameters.

When strict quality of service (QoS) guarantees are not possible, a way in which applications can take advantage of the available CPU capacity is by adapting its processing to system load. This is possible due to the nature of multimedia data; in many cases, it may be better to temporarily degrade a stream rather than completely interrupting the presentation. In such cases, multimedia streams' deadlines are not hard deadlines.

In an adaptive multimedia presentation application, continuous media streams and discrete data are treated simultaneously, and there exist some media scaling mechanisms to adapt presentation to system load. Temporal synchronization enforcement mechanisms are needed in order to respect a user-defined quality of service (QoS). In existing multimedia presentation applications, synchronization between data is generally implemented at a user level by the application itself, so the system has no knowledge of the inter-dependency of data. Performance losses arise when an OS scheduling policy does not consider synchronization relations between data streams in an adaptive application [1], because threads may be scheduled to process data out of time, and these data will be furtherly lost due to synchronization adjustment. These losses can hence be avoided if synchronization is taken into account for thread scheduling.

We believe that generic synchronization mechanisms can be implemented at the system level, so programmers need not to manage synchronization enforcement. This can be done by defining a scheduling policy in which decisions depend not only on the time characteristics of a single stream, but on the time dependencies with other streams. Furthermore, the system can take advantage of media scalability for its resource allocation.

2 Scheduling framework

The scheduling policy we propose needs a certain knowledge of stream synchronization. This is only possible if a more general framework is defined, in which the following important aspects are considered:

- a model for application structure and synchronization specification
- a scheduling policy based on synchronization specification and scalability
- a scaling policy

2.1 Application structure

An adapted application structure is needed to take full advantage of hardware capacities, as system overhead needs to be minimized. One way to achieve this is by using threads as a structuring unit, to treat separately different streams and different activities. Our work focuses mainly on scheduling issues, but we have to consider that a certain application structure is used to manage the different data streams.