Integration of QoS Guarantees into SMIL and its Flexible Implementation

Yoshiki Terashima¹, Keichi Yasumoto¹‡, Teruo Higashino¹, Kota Abe¹, Toshio Matsuura¹ and Kenichi Taniguchi¹

¹ Dept. Info. & Math. Sci., Osaka Univ.
² Dept. Info. Proc. & Man.. Shiga Univ.
³ Media Center, Osaka City Univ.

Toyonaka, Osaka 560-8531, JAPAN
² Hikone, Shiga 522-8522, JAPAN
³ Sumiyoshi, Osaka 558-8855, JAPAN

Abstract—In this paper, we propose a flexible implementation technique for QoS control mechanisms where they are implemented as separate modules using the constraint-oriented style of real-time LOTOS. For an application, we define and implement QOS-SMIL with (1) dynamic switching and (2) explicit inter-media synchronization facilities. Experimental results show that the proposed technique has some advantages w.r.t. development cost for QoS control mechanisms.

I. INTRODUCTION

For effective presentation of multi-media objects (e.g., video, image, audio, text and so on), W3C has standardized a multi-media description language called SMIL 1.0 [1]. In distributed multimedia systems where available network/system resources dynamically change, control mechanisms for guaranteeing quality of service [2] are required in general. However, SMIL 1.0 regards such QoS guarantees as an implementation matter and provides no facility for specifying them. Therefore, to apply SMIL 1.0 directly to development of distributed multimedia systems, we need to extend SMIL to be able to specify some kind of QoS requirements and to develop SMIL players with the corresponding QoS control mechanisms.

As candidates to be integrated into SMIL, various kinds of QoS guarantee requirements are considered, for example, dynamic media scaling [3], inter-media synchronization [4], [5], dynamic server selection [6], and so on. When we implement those mechanisms into SMIL players, we have to carefully consider what sub-set of those mechanisms should be implemented and what kind of implementation is selected for each control mechanism, depending on user environments.

In this paper, we propose a flexible implementation technique for QoS control mechanisms for SMIL. In the proposed technique, we use a subclass of E-LOTOS [7] (called real-time LOTOS [8], [9]) as an intermediate language since it can nicely treat timing constraints for actions and synchronization among parallel processes used in SMIL documents. We implement QoS control mechanisms using the constraint-oriented style [10] of real-time LOTOS where a system is composed of a main process (e.g., video/audio playback) and several constraint processes (e.g., congestion detection, media scaling, inter-media synchronization, and so on). Those processes run in parallel satisfying the specified constraints. Using the proposed technique, we can easily select and implement different combinations from various QoS control mechanisms suitable for target environments.

We define QOS-SMIL where two major QoS guarantee statements, (1) a dynamic switching facility among alternative objects, and (2) a precise inter-media synchronization facility, are introduced to a subclass of SMIL 1.0, and apply the proposed technique to its implementation. With our real-time LOTOS compiler [8], [9], QOS-SMIL documents are implemented as executable programs with our real-time thread mechanism [8]. In each generated program, object playback processes are assigned to the corresponding real-time threads, and all the threads are scheduled in EDF (Earliest Deadline First) manner.

Although it is known that we need some overhead of runtime synchronization in the constraint-oriented style, we show through some experiments that the generated programs could achieve at least 80% of execution efficiency and about half of development cost (w.r.t. description amount), compared with programs directly coded by C with the same real-time thread mechanism.

II. QoS-SMIL

In QOS-SMIL, we introduce (1) dswitch element for dynamic switching among alternative objects, (2) pri attribute for specifying priority to each object, and (3) maxskew attribute for precise inter-media synchronization.

(1) dynamic switching among alternative objects

dswitch element enables dynamic selection and playbacks among specified alternative objects depending on variation of available system/network resources. Major difference from switch element of SMIL 1.0 is that the continuity of scenes is preserved before and after the switching.

In order to adapt to both system and network load variation, the following types of alternative objects can be specified in dswitch element: (i) the same content objects with different temporal resolution, (ii) the same content objects encoded with different quality (w.r.t. picture size, color depth and so on), and (iii) the same content objects specified by different URLs. In dswitch element, alternative objects are supposed to appear in decreasing order of required resource amount. We show an example of dswitch element in Table I. In the table, the first element specifies the full-motion playback of video.mpg, and the second one specifies the playback of video.mpg at 50% reduced time resolution as the alternative object of the first one. The third one and the last one specify the playback of video.half.mpg (e.g., encoded with half picture size of video.mpg) and text.txt (e.g., alternative text), respectively, as the lower quality alternative objects.

(2) priority among multiple concurrent objects

For media scaling, we have introduced pri attribute so that quality of object playbacks with lower priority is reduced prior to objects with higher priority. For each alternative object in dswitch element, a positive integer number is given as priority. The number 1 means the highest priority, and the larger number has the lower priority. The priority values are compared among parallel object playbacks specified in par element. We show an example in Table II. In the table, initially two objects: A (pri=100) and B (pri=70) are played back in parallel. When system load becomes high, A (pri=100) which has lower priority than B is switched to its alternatives in order of A tempcrop=33 (pri=90), A tempcrop=66 (pri=80) and A tempcrop=86. If still overloaded, then B is similarly switched to its lower quality alternatives in order of B tempcrop=33 (pri=70),... and B tempcrop=86. If system/network is recovered from congestion, the above switching mechanism is carried out in reverse order.

(3) precise inter-media synchronization

With maxskew attribute, we can specify acceptable maximum skew deviation in inter-media synchronization of parallel object playbacks in par element (e.g., <par maxskew="0.2s">

III. FLEXIBLE IMPLEMENTATION OF QoS-SMIL

A. Outline of real-time LOTOS

In LOTOS [11], a system is modeled by a set of several concurrent processes where each process’s behavior is defined by sequences of events (input/output action at an interaction point called gate). ax: int and !b: Ex denote an event to input an integer value to variable x at gate a and an event to output the value of Ex to gate b, respectively. In order to specify temporal ordering of events, the action-prefix (a\[ \])
B), the disabling operator (B >>> B), and the enabling operator (B > B) can be specified between any two behavior expressions B. Also, [bool]x -> B can be specified as the condition to execute B if boolean expression bool holds. With the parallel operator with synchronization gates (B || gl) B, we can specify multiple concurrent processes to execute some events and exchange data values in synchronization with each other through gates in gl (called multi-way synchronization). Real-time LOTOS [8], [9] is a variation of LOTOS where some new operators from E-LOTOS [7] such as timing restriction @?T [constraint(t)], wait(d), iteration loop B endloop and while (Exp) B endwhile, write-many variables var . . . endvar, value substitution without gates ?v:=Exp are added to LOTOS (here, constraint(t) is restricted to the form of time interval, i.e., C1 ≤ t ≤ C2).

### B. Implementation of basic facilities of SMIL

We implement a QOS-SMIL document as the corresponding real-time LOTOS specification composed of two parts: (1) MainPart specifying main behavior consisting of event sequences for object playbacks, and (2) TimingPart consisting of multiple constraint processes which specify execution order and timing restrictions for the events in MainPart. This style is called constraint oriented style, and is known as a useful method to design and maintain system specifications [10].

Below, we present an implementation technique using an example of video playback process Play(Obj1) (playback processes for audio and other media can be implemented in the similar technique). We have implemented primitives for processing each unit of video/audio data [9]. When we assign each primitive to an event, MainPart of a video playback is represented by the following event sequence as shown in Fig.1: (1) reading a unit of frame data, (2) its decoding, and (3) drawing the decoded frame on a window (v!Draw) or (4) skipping (v!Skip). Note that v!Skip corresponds to interruption for the event sequence of (1), (2) and (3).

TimingPart gives timing constraints to behavior of MainPart. In Fig.1, TimingPart specifies event v!Draw to be executed every P time unit by v!Draw@?t [t<P] (P is a time period for processing each video frame), or to be interrupted by event v!Skip@?t [t≤P] when the elapsed time exceeds P.

By specifying synchronization on gate v between the above two processes MainPart and TimingPart, the event sequence of reading, decoding and drawing of each video frame is executed in every P time units. If the sequence cannot be executed in P time units, v!Skip interrupts it (i.e., video frame is not drawn in the period). In Fig.1, c!Start, c!End and c!Intr are used to control starting, ending and interruption of video playback. r!Success and r!Miss are used for congestion detection.

By using the above technique, we can easily add new control mechanisms as constraints like TimingPart. Here, note that the important point in this implementation is that we need not modify MainPart in most cases when we add new control mechanisms.

### C. Implementation of QoS specific facilities

Due to space limitation, here we only present the implementation technique for dswitch element (see [12] for implementation techniques of other elements).

For implementation of dswitch element, we need (1) a congestion detection mechanism and (2) a seamless switching mechanism between alternative objects.

#### (1) Congestion detection

For congestion detection in end systems, we have several methodologies: to obtain CPU load by OS system call; or to use monitoring agents which collect some information from playback processes. The proposed technique is based on the latter. Basically, we let a monitoring agent to periodically watch in what rate each playback process completes its job before its deadline. When the rates are lower than a threshold in some processes, we consider that congestion has occurred. We show an example description of such a monitoring agent in Table III where T denotes the monitoring period.

```
Process Monitor checks whether each playback process could complete a frame processing or not by r!1id!Success or r!1id!Miss and counts and keeps the numbers of complete processes and missing ones (The function Add(x,i,val) increments i-th element of the array x by val). After monitoring period T, Monitor calculates the maximum missing rate among all objects (MaxMissRate(suc,mis)) calculates and returns the value). Here, we have described the above process so as to report (i) congestion (s!Congestion) when the maximum missing rate is greater than 30%, (ii) acceptable situation (s!Tolerable) when the rate is between 10% and 30%, or (iii) stable situation (s!Stable) when the rate is below 10%.

For objects specified by URLs, we need to detect network congestion. The similar idea can be used, e.g., calculating available bandwidth by watching mean packet receiving rate.

#### (2) Dynamic switching among alternative objects

By definition of QOS-SMIL, we select the object with the lowest priority among concurrent objects and switch it to the lower alternative object specified in the next line of dswitch element. When congestion st.ill exists after a while (the monitoring agent reports s!Congestion, T time units after), the object with the second lowest priority is switched to its alternative. If congestion still exists, we perform the object playback which was lowered previously should be recovered. Although there are various criteria for concluding that congestion was over, here we adopt the policy that congestion was over when the monitoring agent had reported & s!Stable successively (k is a threshold). Thus, we have implemented process Control to do the above job for each object playback process (see Fig.2).

Concerning about media scaling, when the switching is just to reduce the temporal resolution of an object, we define constraints for several levels of timing resolution in QualityDef.Part which is part of TimingPart for each video playback process (the cases for reducing spatial resolution and for changing servers, see [12]). Fig. 2 shows an example of tempcprop=50 which executes v!Draw and v!Skip alternately. When congestion occurs, the appropriate level is...
The proposed technique may take some synchronization overhead among processes caused by the constraint oriented style. In order to evaluate the trade-off between execution efficiency and development cost, we have carried out some experiments in [12] (RedHat Linux 5.2 on Pentium III 500MHz CPU with 128Mbyte memory). As a result, we have confirmed that (1) programs developed with the proposed technique consume at most 20% more processor power (10% for synchronization overhead and another 10% for compiler overhead) than the programs coded by C, that (2) the description amount with our technique was about half of the C programs in terms of the number of statements without definition part for processes/functions and variables (here we used the same multi-media primitives and the same real-time thread library), and that (3) the modification amount for the inter-media synchronization mechanism with our technique was much less than the case using C (for the details, see [8], [9]).

In [12], we have confirmed that the maximum skew deviation between two parallel video playbacks can be kept within about 20ms under normal UNIX systems. Here, we have evaluated effect of our inter-media synchronization control mechanism with dynamic media scaling. We used two objects $Obj_1$ and $Obj_2$ each encoded with 15fps where the temporal resolution of $Obj_1$ can be reduced to 7.5fps, 5fps and 2.5fps by `daws1ach` element. In the experiment, we gave some load to the system by executing another program after 10 seconds from the beginning of object playbacks. The result is shown in Fig. 3. Fig. 3(a) shows skew deviation between the time when picture frames of the two videos were drawn, and (b) shows deviation of their frame rates. In (b), at the 10 second point, frame rates of both $Obj_1$ and $Obj_2$ dropped down to about 10 fps by the given load. After that, the frame rate of $Obj_1$ was reduced to 7.5 fps and then to 5 fps gradually, and the frame rate of $Obj_2$ was recovered to the original 15 fps. Fig. 3(a) shows that the skew deviation between two videos could be controlled within the specified value 0.08s during such dynamic media scaling.

V. CONCLUSION

In the proposed technique, using the constraint oriented style of real-time LOTOS, we can compose a system of two separate parts: a main process for object playback processes and several constraint processes for QoS control mechanisms. So, we can easily modify or replace each QoS control mechanism depending on target environments. Experimental results show that programs implemented by our technique have almost as good performance as the programs coded in C (with the same libraries), and that the development cost w.r.t. description amount is much less than the other. As a total, the proposed method seems especially useful for rapid prototyping of QoS control mechanisms.

REFERENCES