TMO Structuring of a Networked System for Seamless Streaming and Tiled Display of High-Definition Movies

Sheng Liu[1], K.H. (Kane) Kim[1], Sung-Jin Kim[1], Zhen Zhang[1], Jongho Nang[2], Ki-Seok Choi[2], and Yongbin Kang[3]

[1] Dream Lab, EECS dept., University of California, Irvine
Irvine,CA 92697, USA
{shengl, khkim, sungjink, zhen}@uci.edu
[2] Sogang University, Seoul, Korea
{jhnang}@sogang.ac.kr
[3] Institute for Graphic Interface, Seoul, Korea

Abstract. This paper presents a global-time based approach for realizing high-definition video streaming and highly synchronous tiled display on multiple PC-oriented display nodes. The challenge is to minimize distortion of the temporal relationship among the fragments of video frames played across the display devices. The distortion arises due to the jitter in message transmission delay and the considerably autonomous operations of display nodes. The global-time based coordination approach looked promising as a cost-effective approach for facilitating highly synchronous tiled display and minimizing the collisions between data transmission activities and tile preparation and display activities. The Time-triggered Message-triggered Object (TMO) programming tool-kit, which enables construction of scalable distributed real-time computing programs in the form of networks of high-level, easily analyzable, real-time objects was used because it facilitates efficient practice of the global-time based approach. The results of an experimentation of the approach are also presented.

Keywords: real time, global time, multimedia, HD video streaming, tiled display, jitter, TMO, time triggered, message, object, middleware, distributed, programming, intra-stream synchronization.

1 Introduction

The approach of utilizing PCs, each equipped with a display device, to achieve a large tiled display of a very high-resolution image has been gaining popularity in the past decade. Initially, systems capable of tiled display of static images were built but lately, attempts to handle video-movies started. Video frames forming a movie need to be streamed in timely manners and displayed across multiple display nodes in sufficiently synchronous manners. To achieve high-level *quality of services (QoSs)*, high-precision synchronizations of video streams across multiple display nodes is

important. More precisely, the accurate maintenance of the temporal relationship among media units (MUs) such as audio packets and segments of video frames without the loss of RT performance is required [1, 2, 16]. In other worlds, all segments from the same video frame must be displayed "with a high degree of synchrony", i.e., as closely in the time domain as possible. Each video frame, composed of the tile-segments, and corresponding audio packets must also be presented with a high degree of synchrony. Another challenge is to reduce the amount of efforts for designing and implementing the complex application that performs RT video streaming from the source such as a camera, Internet, and a hard disk to the multiple display nodes and high-quality tiled display of a movie.

In this paper, the principle of *global-time-based coordination of distributed actions (TCoDA)* [14] is exploited to form a fundamental and promising approach for meeting the aforementioned QoS (quality-of-service) requirements. As an approach for significantly reducing the design and implementation efforts from that required under the widely practiced low-level programming schemes involving manipulations of threads, thread-priorities, and sockets, the *Time-triggered Message-triggered Object (TMO)* programming scheme was adopted. The scheme backed by an appropriate tool-set (http://dream.eng.uci.edu/TMOdownload/), facilitates easy practice of object and component-oriented (OCO) real-time (RT) distributed programming [3, 5, 6, 7, 8, 9].

To make this paper self-contained, a brief overview of the TMO programming scheme and DirectShow are introduced in Section 2. The TCoDA-based approach for realizing high-quality real-time video streaming services over a tiled display system is discussed in Section 3. The TMO-based implementation for achieving the minimal loss of the temporal relationship among the video data units at all sink nodes are presented in Section 4. Performance measurements discussed in Section 5 have shown that TCoDA is a fundamental and promising approach for distributed real-time multimedia computing. The paper concludes in Section 6.

2 Backgroud

2.1 TMO Scheme

TMO is a natural, syntactically minor, and semantically powerful extension of conventional object structure. As depicted in Fig. 1, the basic TMO structure consists of four parts:

ODS-sec: Object-data-store section. This section contains the data-container variables shared among methods of a TMO. Variables are grouped into ODS segments (ODSSs) which are the units that can be locked for exclusive use by a TMO method in execution. Access rights of TMO methods for ODSSs are explicitly specified and the execution engine analyzes them to exploit maximal concurrency.

EAC-sec: Environment access capability section. These "gate objects" provide efficient call-paths to remote object methods, real-time multicast and memory replication channels (RMMCs) (Kim et al. 2005), and I/O device interfaces.

SpM-sec: Spontaneous method section. These are time-triggered methods that become alive at specified times.

SvM-sec: Service method section. These provide service methods which can be called by other TMOs.

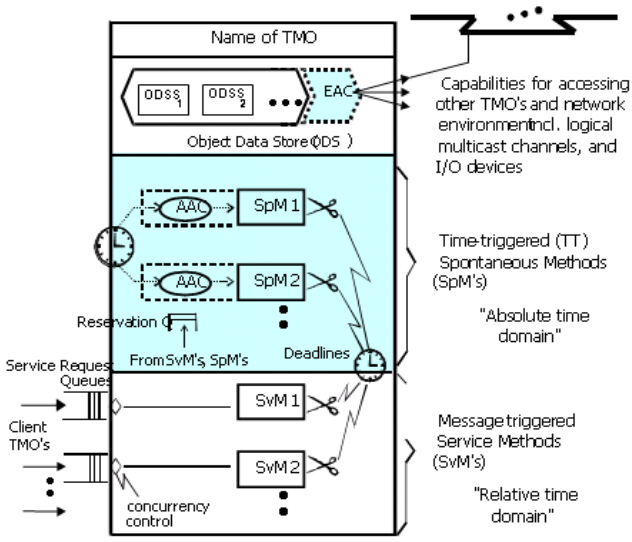


Fig. 1. Basic TMO Structure ([5])

Major features are summarized below.

1) Distributed computing component: The TMO is a distributed computing component and thus TMOs distributed over multiple nodes may interact via remote method calls. To maximize the concurrency in execution of client methods in one node and server methods in the same node or different nodes, client methods are allowed to make non-blocking service requests to service methods. In addition, TMOs can interact by exchange of messages over Real-time Multicast and Memory-Replication Channels (RMMCs) [12]. In any place within a TMO, all time references are global time references except where specified explicitly and differenetly.
2) Clear separation between two types of methods: The TMO may contain two types of methods, time triggered (TT) methods (spontaneous methods or SpMs), which are clearly separated from the conventional service methods (SvMs). The SpM executions are triggered when the RT clock reaches time values determined at the design time. On the contrary, SvM executions are triggered by calls from clients that are transmitted by the execution engine in the form of service request messages. Moreover, actions to be taken at real times, which can be determined at the design time, can appear only in SpMs.

   Triggering times for SpMs must be fully specified as constants during the design time. Those RT constants as well as related guaranteed completion times (GCTs) of the SpM appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is “for t = from 10 am to 10:50 am every 30 min start-during (t , t + 5 min) finish-by t +10 min” which has the same effect as {“start-during (10 am, 10:05 am) finish-by 10:10 am”, “start-during (10:30 am, 10:35 am) finish-by 10:40 am”}.

Executions of SpMs cannot be disturbed by the executions of SvMs because of the execution rule adopted and called the basic concurrency constraint (BCC).

2.2 RMMC

In the TMO programming model, the RMMC scheme is an alternative to the remote method invocation for facilitating interactions among TMOs. Use of RMMCs tends to lead to better efficiency than the use of traditional remote method invocations does in many applications, especially in the area of distributed multimedia applications which involve frequent delivery of the same data to more than two participants housed in different nodes.

In order for methods in a TMO to send and receive messages over RMMCs, the TMO must contain access gates for the RMMCs in its ODS, i.e., as its data members. For example, "access gates" for two RMMCs, RMMC1 and RMMC2, can be declared as data members of each of the three remotely cooperating RT objects, TMO1, TMO2, and TMO3, during the design time. Once TMO1 sends a message over RMMC1, then the message will be delivered to the buffer allocated inside the execution engines for each of the three RT objects. Later during their execution, certain methods in TMO2 and TMO3 can pick up those messages by sending the requests through their RMMC1 gates to their execution engines. An RMMC can be implemented over point-to-point networks as well as over broadcast-enabled bus networks.

2.3 Non-blocking Buffer (NBB)

The NBB mechanism [18] facilitates communication of event messages from a producer to a consumer without causing any party to experience blocking. Therefore, its application scope includes all conceivable producer-consumer situations. Experiments involving application of this mechanism in building middleware as well as real-time application software confirmed the usefulness of the NBB mechanism.

The producer thread, PROD, owns the circular buffer and can write into the buffer at any time without experiencing blocking and thus is a non-blocking writer of the buffer. There are also two counters: the update counter (UC) and the acknowledgment counter (AC), also called the ack-counter. The two counters are used in ways which ensure that PROD and CONS always access different slots in the circular buffer.

2.4 DirectShow

DirectShow is a multimedia application library produced by Microsoft. It provides a set of APIs which defines various operations for multimedia tasks. Within the framework of DirectShow, the processing of a multimedia task can be divided into a set of steps such as reading the media source, encode/decode, and playback. Each of these steps can be accomplished by a module called a filter. Filters can be connected together through their “input/output” pins so as to form a filter graph that can perform a target multimedia task.

3 Global-Time-based Approach for HD Video Streaming and Tiled Display

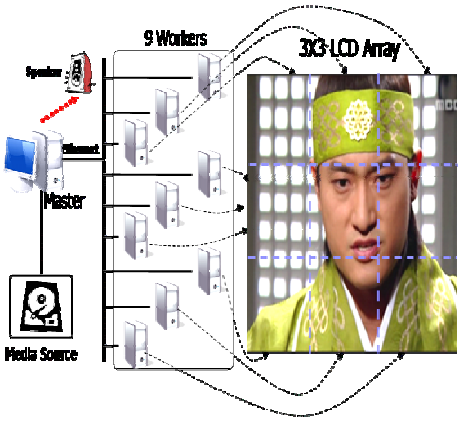
A tiled display system [4, 15, 17] is a cost-effective approach for realizing a large display wall. The combined resolution of a tiled display can easily surpass the resolution of a HD video stream. To utilize a tiled display as an RT HD video stream display, the underlying software system must be able to control the graphic display output of each node to be synchronized so that the user experience a temporally consistent and contextually unified video stream.

Due to the jitters in the network, each video frame in a stream may arrive or may be picked up in different nodes at different times. Also the decompression time taken may be different in different display nodes. Such discrepancies among display nodes may introduce out-of-sync display of different segment of the same video frame. One may make all display nodes wait for a sync message after each decompression process and update the display screen upon receiving the message. However this sync message may arrive or may be picked up in different nodes at a noticeably different times.

With the presence of the global time base of a sub-millisecond precision, one can explain the TCoDA [14], i.e., design all display nodes to render their parts of the video frame at the same instance of global time, e.g., at the target display time = video frame generation + display delay constant. Different applications of the approach have been studied and proof-of-concept systems have been developed [10, 11, 13].

4. TMO-based HD Tiled Display System

4.1. System Architecture



A high-definition video streaming service on a tiled display system consists of one master node and multiple worker nodes, all of which are connected through a LAN. The function of the master node is to retrieve encoded multimedia data from a media source and stream them to the workers with playback timing information embedded. It also takes the responsibility of audio stream playback. The function of a worker is to receive encoded video frames from the communication channel, decode

them, and display its assigned tile-segments of video frame that are located on the basis of its unique Worker ID (WID). Fig. 2 illustrates the system architecture.

4.2. Design of the Master Node

The master node includes two software modules: the master filter graph and the master TMO, which are shown in Fig. 3. The master filter graph is an application program built by interconnecting DirectShow library module provided by Microsoft. Each library module is called a filter. This application runs whenever TMOSM running at the top priority-level yields a time-slice of the machine to non-TMO software, which occurs every 3rd time-slice arrives. In the master filer graph, a built-in *source filter* is used for retrieving compressed audio packets and video frames from a media source. Such a media source can be a media file in the local disk or an URL containing a media stream. In the case of audio packets, they are forwarded by the source filter to a local *audio decoder filter* for decoding since the audio stream will be played back in the master node. A customized *grabber filter* is used to grab uncompressed audio packets out of the master filter graph and put them into an output queue. Another grabber filter is used to pull compressed video frames out of the master filter graph into a separate output queue. Inside a *grabber filter*, a callback function is invoked whenever a media frame becomes available from the output pin of an upstream filter. The *audio grabber filter* is connected to the upstream *audio decoder filter* to grab uncompressed audio packets for local playback while the *video grabber filter* is directly connected to the *source filter* to grab compressed video frames for multicast delivery. The output pin of each grabber filter is connected with the input pin of a customized *flow control filter*, which controls the retrieving speed of the source filter (e.g. 30 frames per second); otherwise, the source filter reads media units from the media source as quickly as possible, which necessitates a large buffer to hold all retrieved media data before they can be played back or delivered to worker nodes.

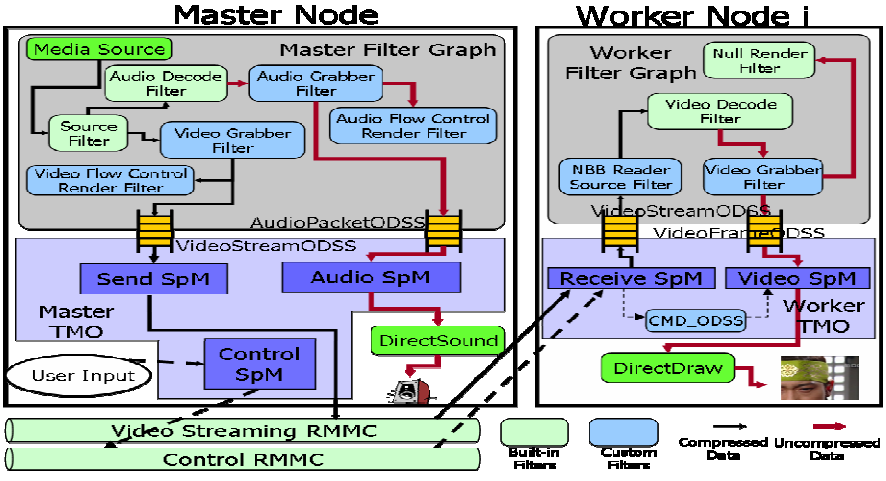


Fig. 3. System Components

Another software module in the master node is the Master TMO, which handles the task of playing back the audio stream and multicasting video stream to all worker nodes. Audio packets and video frames are obtained from the grabber filters in the maser filter graph and need to be picked up by the SpMs in the Master TMO. The grabber filters and the SpMs run in the different thread contexts. An NBB is used to let the grabber filters deposit media data and let the Master TMO SpMs in non-blocking manners. Each NBB is wrapped as a special ODSS in the Master TMO, and its handle is sent to the master filter graph during the system initialization. Two types of ODSS's, namely AudioPacketODSS and VideoStreamODSS are constructed. The *audio grabber filter*, as the writer of the audio NBB, inserts uncompressed audio packets into AudioPacketODSS. Similarly, VideoStreamODSS encapsulates an NBB for holding compressed video frames supplied by the *video grabber filter*.

Besides two ODSS's, the Master TMO contains three SpMs. Audio SpM is used to playback the audio stream. It periodically reads an uncompressed audio packet from AudioPacketODSS and plays it back through Win32 DirectSound APIs. Since an audio packet is played back at the target instant of global time at the beginning of each round of Audio SpM, the intra-stream synchronization jitter is minimized.

Send SpM periodically reads a video frame out of VideoStreamODSS and multicasts it to all workers. In our design, its iteration rate is set to be the same as the frame rate of the media stream being rendered. For example, it runs every 33ms, which means it sends 30 video frames per second. To facilitate video frames to all workers, a video streaming RMMC is constructed. A gate to this RMMC is instantiated during the initialization of the Master TMO. When Send SpM obtains a video frame, it invokes RMMC API Announce() to multicast it out. Since the sizes of compressed video frames are not a constant while an RMMC requires each packet be of fixed size, a video frame needs to be packetized into RMMC packets before being sent out.

The third SpM, Control SpM, runs at the lowest frequency to take user's commands and multicast corresponding control messages, such as "PLAY" and "STOP", to all workers.

4.3. Design of Worker Nodes

Receive SpM in a worker node receives RMMC packets from the video streaming RMMC by calling RMMC API NonBlockingReceiver(), and assembles them if they belong to the same video frame. Then, a complete video frame is inserted into VideoStreamODSS, which is of the NBB type.

Video frames in VideoStreamODSS are read by a customized *source filter*, NBB Reader Source Filter, in the worker filter graph. Its output pin is connected to the input pin of the *video decoder filter*. After being decoded, an uncompressed video frame is sent to the *video grabber filter*. Through the callback function inside the video grabber filter, an uncompressed video frame, the part of a video frame corresponding to the worker's WID, is inserted into the VideoFrameODSS which is of the NBB type. A *null render filter* is added at the end of the worker filter graph by being connected to the output pin of the *video grabber filter* to complete the connection of the worker filter graph. Note that there is no need to use a *flow control*

filter on the worker side since the master node has already controlled the media delivery rate.

Similar to Audio SpM, Video SpM periodically gets a video-fame-fragment from VideoFrameODSS and invokes Win32 DirectDraw API to play it back.

4.4. Synchronous Play of the Video Stream in All Worker Nodes

Synchronous play of video stream in all worker nodes is subject to two requirements:

- The play of the video stream starts with a minimal deviation in the time dimension in all worker nodes.
- Video frames are played back with the same rate in all worker nodes.

As mentioned in Section 4.3, periodical executions of Video SpMs in the Worker TMOs meet the second requirement.

Both the master and workers start with certain internal initialization such as starting TMO engine, filter graph construction, initialization of audio/video devices, and registration of SpM and the TMO.

Once these internal initializations are done, Control SpM in the master node waits for user's inputs. After selecting a media source, the user inputs a "PLAY" command. When Control SpM receives this command, it sends a "Play" message to all worker nodes through a control RMMC. Thereafter, it starts the master filter graph and cosequentially, video frames are fetched from the media source and sent to worker nodes through the video streaming RMMC. When a worker node receives the "Play" message through Receive SpM, it starts the worker filter graph and begins to receive video frames from the master node, decode, and buffer them, but not play them back. The purpose for buffering is obvious, to smooth the transmission jitter. The "Play" message also contains an *Initial Play Time (IPT)* to dictate a moment at which all workers shall begin video playback. The value of IPT is saved into the CMD_ODSS in the worker TMO. Video SpM reads its value from the CMD_ODSS and compares it with the current time at the beginning of each round. If IPT is larger than the current time, it fetches the first video frame from VideoFrameODSS to start play.

One critical point is that the Master TMO needs to choose an appropriate IPT before sending out the "PLAY" message so that the Video SpMs on all workers may begin video-playback with a minimal deviation in the time dimension. It means that at least one frame-fragment should be available in the buffer within VideoFrameODSS when IPT arrives. Hence, the duration D from the moment at which a "PLAY" commend is issued, which is denoted as T, to IPT should be no less than the duration from T to the time at which the first frame-fragment is deposited into the buffer within VideoFrameODSS. To tolerate transmission jitters of video frames, D can be increased so that several video frame-fragments buffer within VideoFrameODSS when IPT arrives although it will accompany the cost of larger latency.

$$IPT = T + D$$

To play the first frame-fragment exactly on IPT, one straightforward approach is to let Video SpM continuously compare the current time with IPT. When IPT arrives, the first frame-fragment is played back. However, this polling approach needs to occupy and consume CPU resource extensively only for the time comparisons. A

more cost-effecitive approach is to set IPT to the earliest starting time (EST) of the earliest iteration of Video SpM that starts after (T + D) and then to check at the beginning of each iteration of Video SpM if IPT is already past or not. If so, the first frame-fragment will be played immediately. Otherwise, no playback will occur during the current iteration of Video SpM. IPT is thus:

$$IPT = EST + \{ceiling[(T + D) / P]\} * P .$$

where P is the period of Video SpM. In each of the subsequent iterations, a video-frame-fragment is again played at the beginning.

5. Performance Measurement

We ran our tiled display on a 3x3 LCD array. The configuration of each node is, Pentium 2.4G CPU, 512M memory, and Windows XP SP2 OS. Fig. 4 gives a snapshot of our demo.

Fig. 4. Snapshot of Tiled Display on LCD Array

Two performance attributes were measured. We first measured the difference of playback time across all worker nodes. In particular, a timestamp is taken each time DirectDraw is about to be called to display a video-frame-fragment. Then the playback time difference for each video frame is the maximum difference across all workers. Fig. 5 shows the difference which can be regarded as a measure of how well the workers are synchronized in playing the video. From the figure we can see that, workers' playback times can be pretty well synchronized for the given application. The playback time difference was mostly less than 8ms and did not exceed.

We also measured the latency from the time when a video frame was first extracted from the media source by the master node to the time when a fragment of the video frame was played back by a worker node. On the master node side, a timestamp is taken whenever a video frame is grabbed by the video grabber filter. On the other hand, for the worker node, a timestamp is taken every time a video-frame-fragment is to be drawn on the screen. The difference of the two timestamps corresponding to the same video frame can be seen as the delay for processing the video frame in the system. Fig. 6 shows that the latency is about the same for different cases of video frames despite the fact that frames of different sizes may take different amount of time in transmission and decoding.

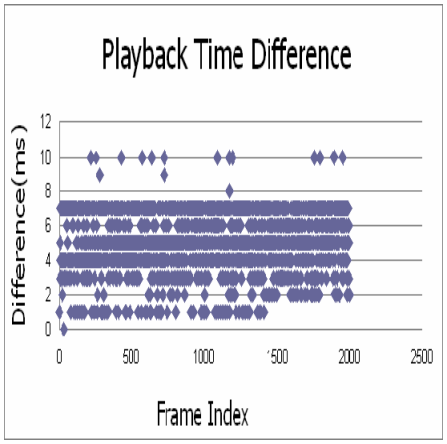


Fig.5. Playback Time Difference

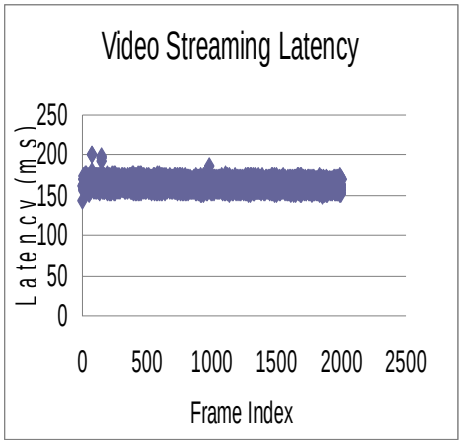


Fig. 6. Transmission Latency of Video Stream

6. Conclusion

Through construction of a global-time-based TMO network, a high-quality high-definition tiled display system was realized. This application case is one demonstration of the global-time-based coordination of distributed actions (TCoDA) as a fundamental and promising approach for distributed multimedia applications. It was also a demonstration that the TMO scheme and its tool-kit enabling relatively easy high-level programming of manipulations of global time-stamps and timely processing of video and other multimedia data reduce the amount of efforts for design and implementation of complex distributed multimedia applications. Our performance measurements have indicated that the TCoDA approach realizes the minimal play jitter in streaming and play of video data.

Acknowledgment:

The work reported here was supported in part by the NSF under Grant Numbers 03-26606 (ITR) and 05-24050 (CNS) and under Cooperative Agreement ANI-0225642 to the University of California, San Diego for "The OptIPuter". No part of this paper represents the views and opinions of any of the sponsors mentioned above.

References

1. J. Ayars, and et. a.l: Synchronized Multimedia Integration Language (SMIL 2.0). In: W3C Recommendation (2001), http://www.w3.org/TR/2001/REC-smil20-20010807

2. Blakowski, G. and Steinmetz, R.: A Media Synchronization Survey: Reference Model, Specification, and Case Studies. IEEE Journal of selected areas in communications, Vol. 14, No. 1, pp. 5-35 (1996)
3. Gimenez, G., and Kim, K.H.: A Windows CE Implementation of a Middleware Architecture Supporting Time-Triggered Message Triggered Objects. In: Proc. IEEE CS Computer Software & Applications Conf., Chicago, IL, pp. 181-189 (2001)
4. HIPerWall, http://hiperwall.calit2.uci.edu
5. Kim, K.H.: Object Structures for Real-Time Systems and Simulators: IEEE Computer, Vol. 30, No.8, pp. 62-70 (1997)
6. Kim, K.H. Ishida, Masaki, and Liu, Juqiang: An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation. In: 2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing, St. Malo, France, pp.54-63 (1999)
7. Kim, K.H.: APIs Enabling High-Level Real-Time Distributed Object Programming. IEEE Computer, pp.72-80 (2000)
8. Kim, K.H., Liu, J.Q., Miyazaki, H., and Shokri, E.H.: TMOES: A CORBA Service Middleware Enabling High-Level Real-Time Object Programming. In: IEEE CS 5th Int'l Symp. on Autonomous Decentralized Systems, Dallas, pp. 327-335 (2001)
9. Kim, K.H.: Commanding and Reactive Control of Peripherals in the TMO Programming Scheme. In: 5th IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing, Crystal City, VA, pp.448-456 (2002)
10. Kim, S., Kuester, F., and Kim, K.H.: A Global-timestamp-based Approach to Construct a Real-time Distributed Tiled Display System. In: EIST-2005 pp. 548-554 (2005)
11. Kim, S., Kuester, F., and Kim, K.H.: A Global timestamp-based Approach to Enhanced Data Consistency and Fairness in Collaborative Virtual Environments. ACM Multimedia System, Vol. 10, No. 3, pp. 220-229 (2003)
12. Kim, K. H., Li, Yuqing, Liu, Sheng, Kim, Moon H., Kim, Doo-Hyun: RMMC Programming Model and Support Execution Engine in the TMO Programming Scheme. In: 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Seattle, Washington, pp. 34-43 (2005)
13. Kim, K.H., Liu, S., Kim, M.H., and Kim, D.H.: A Global-Time-Based Approach for High-Quality Real-Time Video Streaming Services. In: 7th IEEE Int'l Symp. on Multimedia, Irvine, CA, pp. 802-810 (2005)
14. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Pub., ISBN: 0-7923-9894-7, Boston (1997)
15. Renambot, L. and Rao, A.and Singh, R. and Jeong, B. and Krishnaprasad,N. and Vishwanath, V. and Chandrasekhar, V. and Schwarz, N. and Spale, A. and Zhang, C. and Goldman, G. and Leigh, J. and Johnson, A.: SAGE: the Scalable Adaptive Graphics Environment. In: Proceedings of the Workshop on Advanced Collaborative Environments (2004)
16. Steinmetz, R. and Engler, C.: Human Perception of Media Synchronization. Technical Report 43.9310, IBM European Networking Center Heidelberg, Heidelberg, Germany (1993)
17. Wallace, G., Anshus, O. J., Bi, P., Chen, H., Chen, Y., Clark, D., Cook, P., Finkelstein, A., Funkhouser, T., Gupta, A., Hibbs, M., Li, K., Liu, Z., Samanta, R., Sukthankar, R., and Troyanskaya, O.: Tools and Applications for Large-scale Display Walls. Computer Graphics and Applications, vol. 25, pp. 24–33 (2005)
18. Kim, K.H., Clomenares, J., and Rim, K.: Efficient Adaptations of the Non-blocking Buffer for event Message Communication. In: 10th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Santorini Island, Greece (2007)