Event-based Programming Structures for Multimedia Information Flows

K. Ravindran[1] and Ali Sabbir[2]

[1] City College of CUNY and Graduate Center,
Department of Computer Science,
Convent Avenue at 138th Street,
New York, NY 10031, USA
ravi@cs.ccny.cuny.edu
[2] CUNY Graduate Center, Computer Science,
365 Fifth Avenue,
New York, NY 10016, USA
asabbir@hotmail.com

Abstract. In this paper, we propose a programming model based on 'timed event dissemination' for structuring a distributed real-time multimedia presentation. In this model, event notifications capture program-generated actions and/or user-level object accesses on a multimedia window. A coherent effect of these actions requires enforcing deadlines on the event processing over prescribed time intervals. To meet this requirement, the paper advocates an integration of the 'flow of time' as part of the semantics of data presentation on a multimedia window. The paper explores a programming paradigm for event processing: *causal ordering* of timed messages, to realize multimedia data presentations. This yields simplicity and uniformity in the programming structure of multimedia applications. The presentation specifications in our model can be easily and accurately mapped onto system-level QOS parameters (such as network delays and play-out buffer delays) for scheduling purposes. This in turn may lead to an optimal use of the system resources by a multimedia presentation protocol. The generality of our event-oriented programming interface also allows reducing the multimedia system development costs through software reuse.

1 Introduction

A real-time presentation system (RPS) collects the data of various source entities, transports the data through the underlying network, and delivers the data at destination entities for consumption [1]. During a processing of data, maintaining the required temporal association between the data of various streams is necessary in the presence of system induced delays and asynchrony (e.g., multimedia synchronization). This is possible, from the RPS perspective, by segmenting the data streams into application perceivable distinct units along the real-time axis and exercising *temporal presentation control* on these data segments at users. In a multimedia document access for example, a user activity may possibly consist

of graphic input from a menu using mouse click to highlight a certain portion of the document and text input from a keyboard and audio input from a voice phone to annotate an update on the document. Here, the text input may need to be presented at a user station after the mouse click within, say, 2-3 seconds (*sec*). Thus a temporal presentation involves extracting the timing relationship between various data segments collected at sources and determining the delivery times of these data segments at receivers for processing.

The above temporal presentation control depicts a *user-level view* of synchronizing the occurrence of multiple types of data. To support this view, the RPS should embody an application-level specification of temporal presentation control requirements on data. Using this specification, an underlying protocol may implement a set of network level and end-system level mechanisms to meet the data presentation requirements.

The presentation control information is typically made available to the RPS in the form of *quality of service* (QOS) parameters that indicate the various temporal characteristics of data presentation. For instance, how long the processing of a data x at a user can be deferred with respect to that of another data y is specifiable as a QOS parameter. The end-system mechanisms deal with generating real-time *presentation schedules* that control the processing times of data units at various system elements in the path to receivers, so that the QOS parameters specified can be met (by buffering and sequencing the data across the transport-application layer interface). If, for instance, the data x is scheduled for processing at time T but arrives at a receiver at time $(T - t)$, it is buffered by the end-system protocol for a duration of t before presentation to the user. As can be seen, the QOS specification and the underlying presentation protocols form essential parts of a RPS.

This paper provides a flexible and canonical specification model for presentation QOS that can be transcribed onto a communication system. The model is based on application specifiable segmentation of data streams, and playing out data segments in a certain order and in a timely manner, based on the temporal relationships between them.

A key feature of our model is the integration of real-time constraints into a *causal ordering* of data segments (or messages) that flow between end-systems to manipulate user-level objects. In this notion, the messages exchanged between various objects flow in a prescribed order and within prescribed intervals inducing 'state changes' in them (e.g., generation of visual cues in human mind by a video clip that annotates a graphics image display). The application-specific enforcement of presentation control allows flexibility in generating play-out schedules of data and optimizes the usage of system resources to application needs. The paper provides methods to derive the system-level QOS requirements from application-specified information (through APIs) and to parameterize the protocol procedures with this QOS.

The paper is organized as follows: Section 2 describes our model of causal ordering based real-time data presentation. Section 3 describes the specification construct that incorporates the presentation model. Section 4 describes

the protocol procedures to derive network QOS parameters from presentation specifications. Section 5 compares our notion of causal ordering with extensions made to Lamport's 'logical clock'-based message ordering. Section 6 concludes the paper.

2 Model of timed data presentation

A real-time data stream may be segmented into application-specific units for end-to-end transport and synchronization (e.g., a voice clip in multimedia lecturing). The presentation-level processing of a data segment (or message) involves the generation and/or dissemination of device-specific information elements carried in the message over a prescribed time interval. In this section, we describe a temporal characterization of the data presentation problem, which will allow us to determine the structure of presentation-level primitives.

2.1 Sensory perception of data

If t'_m and t_m indicate the time of generation of a message m by the data source (i.e., server) and the start time of play-out of m at a receiver respectively, then $t_m \geq t'_m + \overline{D}$, where $\overline{D}$ is the delay suffered by m in various system elements implementing the message path (such as the network and play-out scheduling buffers). Where multiple inter-stream messages are involved, the play-out of m may be additionally delayed pending the play-out of one or more other messages.

Fast responsiveness and improved cohesiveness at user level requires that the effects of m be seen: i) before the deadline imposed by the timeliness requirements of m, and ii) after the persistence effects message(s) prior to m have already ceased. The timeliness condition may be expressed in terms of the maximum allowed *latency* β_m between the generation and dissemination of m:

$$t_m \leq (t'_m + \beta_m). \tag{1}$$

As example, the layout diagram of a construction building may need to be presented within, say, 1-2 *sec* after requesting the multimedia server, with video and voice illustrations of the diagram timed in a way to enable a virtual tour of the building. If we assume that the designer should see a layout display on the local workstation in response to a point-and-click action within 0.5 *sec* and the upstream delay in network paths is 50 *msec*, we can set $\beta_m \approx 0.45$ *sec*. When $\overline{D} < \beta_m$, the presentation of m can start sooner than the deadline set by β_m. The play-out time of message m should satisfy the following relation:

$$t_m \ > \ t_{m_x} + \delta_{m_x} + \zeta_{m_x}, \tag{2}$$

where m depicts the next action upon seeing message m_x, δ_{m_x} is the play-out time of m_x (i.e., the time over which the data units of m_x are delivered to the end-device for processing) and ζ_{m_x} indicates how long the effects of m_x persist in the application. The choice of t_m needs to account for any user-level

'thinking time' required after the sensing of m_x that will allow generation of the context for m. The 'thinking time' requirement imposes a minimum separation between the occurrence of a set of successive messages. The relation (2) models the passage of real-time with intervals of duration large enough to accommodate the data persistence and user-level thinking effects. In an example of displaying text sequences in a workstation window, δ_m may be 2 *sec* and ζ_m may vary, say, from 1 *sec* to 3 *sec*. From equations (1) and (2), the 'thinking time' and the allowed latency of messages may be related as:

$$(t_{m_x} + \delta_{m_x} + \zeta_{m_x}) < (t'_m + \beta_m). \tag{3}$$

Overall, a data stream is representable by a timed sequence of messages $[\cdots, m_x, m, \cdots]$, with the inter-message separation determined by the user-level 'thinking time' required on m and the persistence duration $(\delta_m + \zeta_m)$.

2.2 'real-time persistence' based causal ordering

The temporal relationships among activities on a user-level object prescribe the order and the real-time intervals in which the messages depicting these activities are presented to users. How these relationships are captured and processed by the RPS is the focus of our paper.

Given two messages y and x describing actions on an object, the relation '$y \succ x$' denotes that x should occur after the persistence duration of y has elapsed and within a certain time limit T^{max}. The meaning is that the state change in the object caused by y persists over a real-time interval $[t_y, t_y + \delta_y + \zeta_y]$ providing the context for a state change by x, where $T^{max} \geq [t_x - (t_y + \delta_y + \zeta_y) + \delta_x + \zeta_x]$. In the earlier example, voice clip may (say) be presented 2 *sec* after the icon highlighting, and persist for a 3 *sec* duration. We refer to the relationship $y \succ x$ as 'x causally depends on y', which depicts a schedule $t_x > t_y + \delta_y + \zeta_y$. When neither $x \succ y$ nor $y \succ x$ is specifiable, y and x are said to be concurrent, denoted as $\|\{x, y\}$. Here, it does not matter whether an object executes x followed by y or vice versa, or simultaneously, as long as the actions are executed within the time limit T^{max}, i.e.,

$$\begin{aligned} T^{max} &\geq max(\{(\delta_y + \zeta_y), (t_x - t_y + \delta_x + \zeta_x)\}) \quad \text{for} \quad t_y \leq t_x \\ T^{max} &\geq max(\{(\delta_x + \zeta_x), (t_y - t_x + \delta_y + \zeta_y)\}) \quad \text{for} \quad t_x \leq t_y. \end{aligned}$$

Causal ordering of messages allows solving the data presentation problem in an application-specific manner. The underlying program structure is uniform across applications: namely, the synchronization of user operations on an object by a temporally ordered processing of the messages. For use with RPS however, the current notions of message causality — defined over 'logical time' — need to be extended for integrating the 'flow of real-time'.

2.3 Temporal intervals

A presentation activity can generate multiple messages, such as a video clip, text information and cursor marker, as part of a single update action on a document[3]. Accordingly, we need to treat the various component messages as bundled together by the temporal relationships between them. A temporal interval is the real-time duration over which the effects of all messages generated by a presentation activity can persist, given as T^{max}. An interval basically indicates the granularity of real-time meaningful to the application, and hence constitutes the unit of segmentation of real-time axis. See Figure 1.

Since a message is the basic unit supported by the RPS in terms of which presentation activities are constructed, the ordering relationship among various activities is transcribed directly onto the corresponding messages. In the earlier example, the highlighting mark on the city map and the text and voice annotation by a user are deemed to occur over a single temporal interval. These messages cannot be interleaved with messages of other activities, since user attention on the map pertains to one activity at a time. On the other hand, messages pertaining to activities on unrelated objects can be interleaved in different ways.

Thus, temporal intervals provide an elegant framework for distributed programming

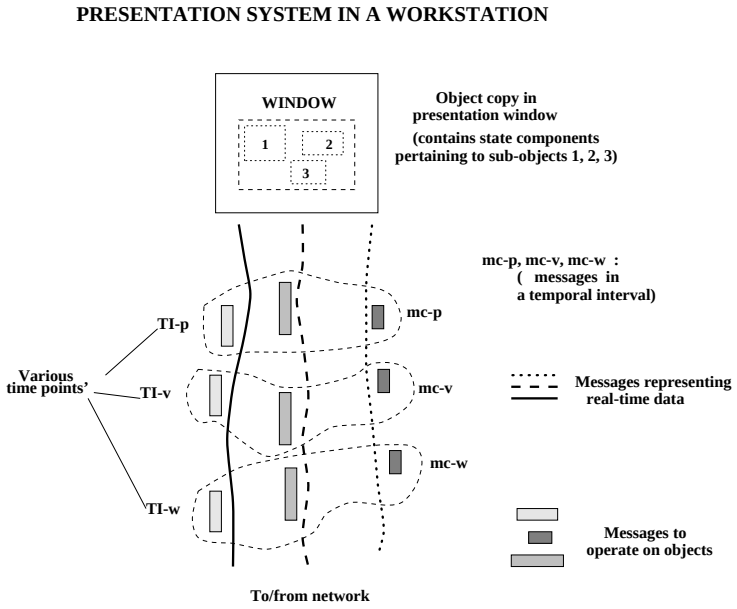


Fig. 1. Flow of messages in data presentation (say, $TI_p < TI_v < TI_w$)

[3] A 'video clip' is treated as a single message for presentation purposes. A lower level system layer may however treat the 'video clip' as consisting of a timed sequence of picture data frames to be delivered at presentation entities.

of functions to access and manipulate user-level objects.

2.4 Synchronization specifications

The temporal ordering relationship between presentation activities on an object determines the extent of concurrency possible in their execution. To reap this concurrency, we need to be able to specify and enforce the required order of occurrence of the corresponding messages, so that they may interleave with one another in many possible ways.

Many existing techniques to determine these requirements are based on time-stamps, whereby a message with a time stamp T is declared to occur after all messages with a time stamp lower than T [2]. These techniques are less suitable for use by RPS, because the human-oriented nature of data presentations allows synchronization constraints that can be weaker [3] than what the RPS is able to infer bottom-up from the message flows incident on it. Precisely extracting the constraints in turn allows exploiting the concurrency more effectively, and hence offers an increased performance potential of the RPS.

Accordingly, we need a programming interface that allows extracting the intended synchronization requirements at the user level. For example, a text annotation to a document update may need to occur after a voice annotation. As another example, a text annotation and a video clip describing a scene can occur in parallel, even though they may be sent one after the other by the source. Such requirements, extracted during execution of a data presentation, can then be used by a protocol in the RPS to determine the scheduling required on messages to move them through various system elements in the path leading to clients (such as network links and play-out buffers).

3 Object-level message ordering

The specification of causal dependency relations between messages needs to be embedded into programming primitives that allow the presentation of messages at user entities. The synchronization related features we embed into the primitives are described below.

3.1 Specification of causal relationships

The application provides a declarative specification of message ordering dependencies. A user entity specifies the temporal ordering of a message z relative to a message y (i.e., $y \succ z$) with the following primitive:

$$((z, p_z),\ Occurs\_After(y, l_z, u_z)),$$

where $p_z = (\delta_z + \zeta_z)$, l_z is the minimum time that should elapse since the occurrence of y, and u_z is the maximum time that can elapse. The parameter l_z is based on the user-level 'thinking time' required when the action y occurs so that

the context for processing a next action z can be generated. The parameter u_z captures the maximum allowed user-level 'responsiveness time' in disseminating the action z once y is seen to have occurred. Given that $u_z > l_z > 0$, the time interval over which z can be played-out is determined from $t_z \in [t_y + \delta_y + l_z, t_y + \delta_y + u_z]$ and δ_z. When $Occurs\_After$(NULL) is specified, z can be processed without any constraint, i.e., immediately upon arrival from the network.

In the earlier example, the concurrent delivery of text and voice annotations to the highlighting mark on a city map, as captured by the causal relation 'highlight $\succ \| \{$text, voice$\}$', may be specified as (times are in seconds):

$$((\text{text}, 5),\ Occurs\_After(\text{highlight}, 2, 4))$$
$$((\text{voice}, 4.5)),\ Occurs\_After(\text{highlight}, 3, 5)),$$

with, say, $\delta_{\text{highlight}} = 1.25$. The time intervals for these messages are given by: $[t_{\text{highlight}} + 1.25 + min(\{2, 3\}),\ t_{\text{highlight}} + 1.25 + max(\{4 + 5, 5 + 4.5\})]$.

The $Occurs\_After$ is basically a programming notation to explicitly construct the causal ordering relation ($\succ$). The causal order constraints are carried in messages for use by the underlying protocols to enforce play-out of the messages in an appropriate sequence and over specified real-time intervals. In this aspect, the persistence parameter p_z may correspond to the 'explicit time duration' allowed in SMIL (the WWW Consortium's Synchronized Multimedia Integration Language [4]).

3.2 Dependency on multiple messages

An extended form of the $Occurs\_After$ construct allows a user to specify complex ordering relationships with AND and OR 'logical connectives' on causally precedent messages. For instance, the AND connective for 2 messages takes the form:

$$((z, p_z),\ Occurs\_After(y_1\ \wedge\ y_2, l, u)),$$

which indicates that z be processed after both y_1 and y_2, with l and u being relative to the latest of y_1 and y_2 (note that $\| \{y_1, y_2\}$). In the previous example, the user may display, say, an annotation on the graphics image to give visual cues for the text and voice information. This may be specified as:

$$((\text{graphics}, 3.25),\ Occurs\_After(\text{text}\ \wedge\ \text{voice},\ 2.5,\ 3.5)).$$

The time interval for occurrence of 'graphics' message is:

$$[max(\{t_{\text{text}} + 5, t_{\text{voice}} + 4.5\}) + 2.5,\ max(\{t_{\text{text}} + 5, t_{\text{voice}} + 4.5\}) + 3.5 + 3.25].$$

See Figure 2 for an illustration. The '$\wedge$' operator linking z to $y_1, y_2, \cdots$ has a stronger semantics than the 'par' construct allowed by SMIL, in that the '$\wedge$' prescribes a concrete parallel composition of operations relative to z.

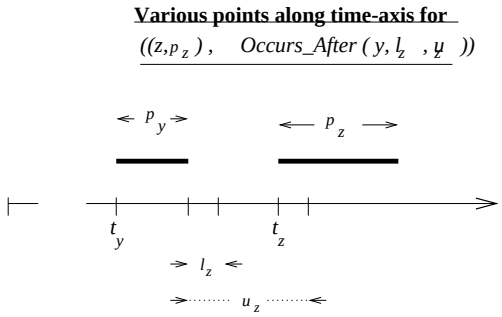
A causal dependency based on OR 'logical connective' is similar to the 'switch' element of SMIL. Again, in a 2-message case, it takes the form:

$$((z, \delta_z),\ Occurs\_After(y_1 \ \vee \ y_2, l, u)),$$

indicating that z be processed after either y_1 or y_2 (or both), with l and u being relative to whichever occurs the earliest during execution. In one scenario for example, user annotation on a graphics image may be generated right after the text or voice information, specified as:

$$((\text{graphics}, 3.25),\ Occurs\_After(\text{text} \ \vee \ \text{voice},\ 2.5,\ 3.5)).$$

The '$\vee$' operator can induce non-determinism in a program execution, with non-reproducible event sequences. The non-determinism may however be restricted



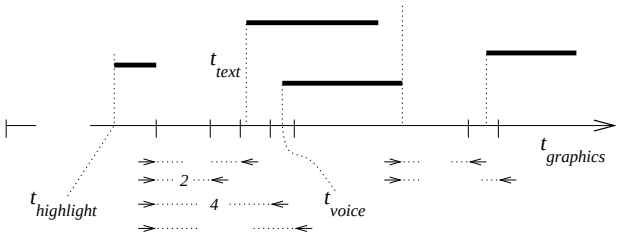


Fig. 2. Illustration of $Occurs\_After$ specifications

to within the current temporal interval.

For brevity, we consider only AND dependencies.

3.3 System delay specifications

The message-level asynchrony captured by causality behavior is incorporated in a 'quality of service' specification QOS_{pres}. The parameters specified in QOS_{pres} may be used to: i) generate a specification of the system delay behavior, and ii) generate a play-out schedule for data segments arriving through system elements.

These functions are incorporated as distinct elements in a synchronization protocol. See Figure 3.

A main feature of delay specification is the allowed end-to-end latency $\overline{D}_q$ on media messages. For interactive multimedia applications (e.g., multi-player video game), $\overline{D}_q$ is set low — say, less than 100 $msec$. For applications with less user-level interactivity (such as 'multimedia lecture presentation' to an audience), $\overline{D}_q$ can be high. Given a set of messages to be presented in a temporal interval, the presentation protocol in the RPS sifts through the $Occurs\_After$ specifications

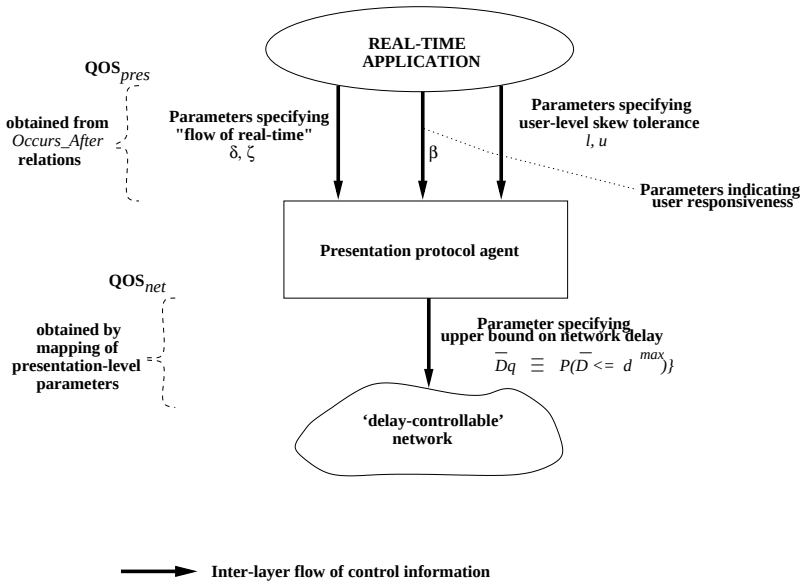


Fig. 3. Mapping of application parameters to network delay behaviors

on various messages for determining $\overline{D}_q$. Since the user-specified parameter $(u_z - l_z)$ indicates the variability allowed in the presentation times of a message z relative to its causally precedent message (i.e., the extent of asynchrony), $\overline{D}_q$ has a direct relationship to this user-level parameter.

A higher value of $\overline{D}_q$ allows sending a message over a transport path with longer message queues and/or containing lower bit-rate communication links. Consider, for example, the sending of a 15 sec MPEG-1 video clip followed by a 10 sec 'thinking time'. The message carrying the video clip consists of 4.05 $mbytes$ of data, generating a bit rate of 2.16 $mbps$ over the 15 sec duration (obtained from trace analysis experiments). When sent over a 1.8 $mbps$ link, the presentation at a receiver has to start about 3 sec later, in comparison to sending over a 2.16 $mbps$ link. So choosing a 1.8 $mbps$ link (against a 2.16 $mbps$ link) is possible only when $\overline{D}_q > 3$ sec. In general, message-level asynchrony captures the user tolerance to latency in data presentations.

$\overline{D}_q$ may often be specified in terms of a bound on the message delays incurred by the network. Another aspect is whether a delay bound is enforced by the network deterministically or probabilistically, with the former requiring a higher allocation of resources in data paths through the network than the latter. In general, the delay specification on the network may be expressed in terms of a delay bound d^{max} in the form:

$$\overline{D}_q \equiv \{(X, P(\overline{D} \leq X))\}_{\forall X \in (d^{min}, d^{max}]}$$

for $d^{max} > d^{min} > 0$, where d^{min} is the minimum delay suffered by messages flowing through system elements and $P(\overline{D} \leq X)$ depicts the probability distribution of the actual message delay incurred. $P(\overline{D} \leq d^{max}) = 1.0$ and $P(\overline{D} \leq d^{max}) < 1.0$ refer to a deterministically delay-bounded path and a probabilistically delay-bounded path respectively. Note that it is the 'data path' from a multimedia server to clients that is subject to delay specification $\overline{D}_q$ (we assume, without loss of generality, that the 'control path' from a client to the server incurs negligible delay).

The media level concurrency prescribed through $Occurs\_After$ relations translates into message-level asynchrony. The latter in turn can be mapped to the allowed variability in network delays incurred on messages (i.e., a specification of $\overline{D}_q$). Refer to Figure 3.

4 Specifying delay controllability

The underlying presentation protocol should take into account the message deadlines prescribed by applications and the resource demands imposed on networks, when specifying a value for d^{max}. The tradeoffs to be considered are as follows (see Figure 4):

- Specifying a large value of d^{max} will reduce the amount of resource demands imposed on the network but may result in some of the messages missing their presentation deadlines;
- Specifying a smaller value of d^{max} will increase the amount of resource demands on the network, but can eliminate the likelihood of messages missing their deadlines.

Also, a higher probability of enforcing a given delay bound places larger resource demands on the network, with a deterministic enforcement of the bound imposing the maximum resource demands[4].

4.1 Presentation skew due to delays

Given a set of messages $\{m\}$, the condition $d^{min} + p_m < \beta_m$ will ensure a non-zero probability of presenting a message m. Assuming that messages are not generated

[4] See [5] for a quantitative study of the underlying message scheduling mechanisms to realize 'parameterizable delay' networks.

ahead of time (as in many live presentation settings), the condition $d^{max} + p_m < \beta_m$ is necessary to avoid a non-presentation of m due to an insufficient life-span of m. As can be seen, the delay specification is relative to the life-span of

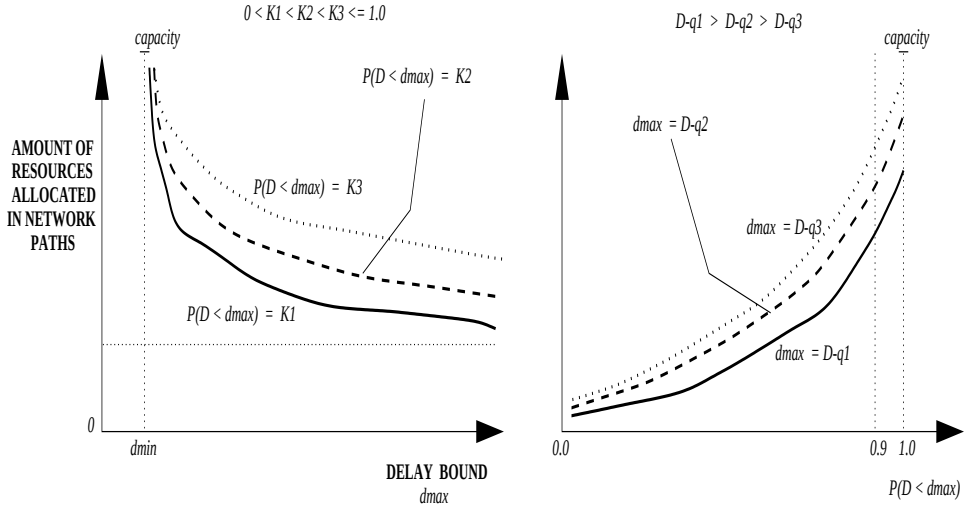


Fig. 4. Network delay behaviors from resource allocation perspective

messages prescribed through β parameters.

Messages can miss their deadlines under two circumstances: when probabilistic delay bounds are specified for network paths (i.e., $P(\overline{D} \leq d^{max}) < 1.0 \; \forall \; d^{max} \in \mathcal{R}^+$) or when lax bounds are specified (i.e., d^{max} is set to a value higher than that prescribed by the β parameters). A higher degree of laxity in message delivery, specified through a lower probability of enforcing a given delay bound, may result in more messages missing their presentation deadlines. This in turn may increase the frequency of *glitches* seen by users when accessing the object. So the level of user tolerance to presentation glitches has a bearing on the extent to which delay bounds need to be enforced. Since a probabilistic delay bound imposes less demand on network resources in comparison to a deterministic bound, a lax user tolerance can be mapped to a reduced resource allocation.

4.2 Handling of missed deadlines

A presentation skew may sometimes exceed the application-level tolerance limits (as set by the u parameter), manifesting as a glitch in the presentation of messages to the application. As an illustrative analysis of the problem of presentation misses, we map the probability distribution governing the network delay behavior on individual messages (in terms of d^{max}) into a probability that one

or more messages from a set of concurrently generated y_j's will meet their deadlines. Figure 5 shows an analysis of the likelihood of successful presentations for a network that enforces probabilistic delay bounds.

Recovery from a glitch may often depend on how long the glitch effects persist

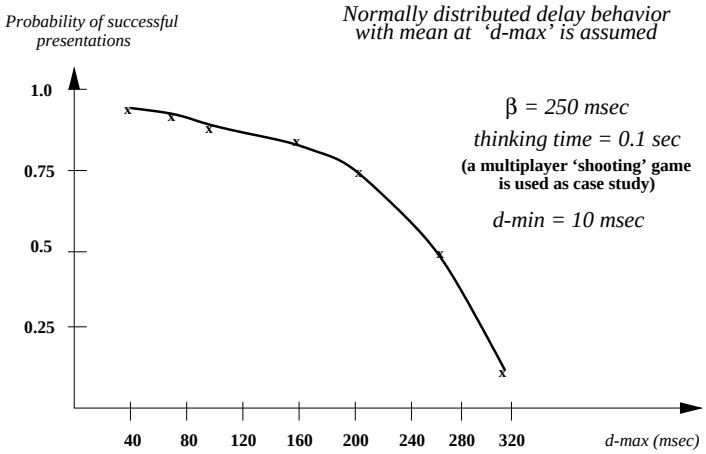


Fig. 5. Presentation probability versus network delay behaviors

and how tolerant the application is to these effects. In a multimedia lecture for example, a glitch is observable as the absence of, say, a video clip to annotate the lecture. The effect of missed visual cues may persist in the minds of the human viewer for a few seconds. When the tolerance limits are exceeded, a recovery from the glitch may manifest in activating an application specified 'exception handler'. The absence of a video clip may however be handled by delivering only the voice clip corresponding to the missed video clip, which may be acceptable due to the slow varying nature of the visual and aural cues at the human viewer [6]. In general, various levels of tolerance to skew among messages are specifiable in QOS_{pres}, with hooks into application-supplied exception handlers.

5 Existing notions of message causality

We argue that the notion of 'message causality' is the right basis for structuring of data presentations in distributed real-time systems for two reasons. First, a synchronization construct based on 'message causality' is readily incorporatable into a distributed programming system since this notion has been well-studied by the distributed systems research community. Second, user-level actions in an interactive setting can be more flexibly expressed through 'cause-and-effect' relationships. In this light, we compare our extended notion of 'message causality' with both the classical approaches that do not support the notion of 'real-time'

and the 'add-on' approaches that are tailor-made to fit 'real-time' into the classical approaches.

5.1 Logical time-stamp based approaches

Currently available causal order primitives [7],[8] infer message causality based on the 'logical time stamps' internally generated by the communication system ('implicit' approach to extracting concurrency in the application). If, for instance, messages m_1 and m_2 are sent in that sequence by a user U, these primitives treat m_2 as causally dependent on m_1. This may not however be the intention of U, such as m_1 and m_2 being concurrent because they are update requests on different (unrelated) documents. Thus existing primitives are at a lower level and do not precisely extract the concurrency in an application, resulting in less message-level asynchrony (i.e., more synchronization latency), in comparison to our 'explicit' approach[5] [9].

Also, the above primitives do not directly incorporate 'real-time' into message-level causality. Even though a 'real-time support' layer can be introduced on top of such causal order based communication systems, this additional layer cannot compensate for the lost concurrency and hence the lost ability to prescribe lax delays when moving the messages through lower level system elements. The only advantage of such a 'real-time' layer will be in lax scheduling of messages at the time of presentation.

5.2 Causality augmented with time deadlines

The[6] notion of 'Δ-causality' proposed in [10] prescribes an upper bound 'Δ' on the delivery time of a message m' relative to its causally precedent message m. This corresponds to our parameter $u_{m'}$ underlying the relation: $m \succ m'$. Though useful for designing real-time transfer protocols in the network with a message delivery bound 'Δ', the 'Δ-causality' notion does not capture the 'flow of real-time' in a distributed computation, because there is no lower bound on the time of delivery of m'. In contrast, our notion allows prescribing a persistence duration p_m of m and a lower bound $l_{m'}$ on the delivery time of m' relative to m, viz., $t_{m'} > (t_m + p_m + l_{m'})$. So our notion is at a higher level, easily employable in distributed programming environments.

The work of J. P. Courtiat and et al [11] uses the notion of 'causality' to realize synchronization. But the specification model treats the 'causality' and 'real-time flow' notions separately, which necessitates more complex programming constructs. However, our work integrates these notions in the specification

[5] An underlying protocol that enforces causal delivery with only implicit information will incur more synchronization latency. In the earlier scenario, buffering of m_2 pending a subsequent arrival of m_1 may increase the time of completion of processing m_1 and m_2, viz., '$> \delta_{m_2} + \delta_{m_1}$' instead of '$> max(\{\delta_{m_1}, \delta_{m_2}\})$'. The performance problem arises due to lack of knowledge in the system that m_1 and m_2 are concurrent.

[6] Our concurrency enhancements are manifestations of multimedia object partitioning in the form of 'spatial subparts' and 'temporal subparts' as allowed in SMIL [4].

model, whereby a concise set of programming constructs suffices to generate presentation schedules. Likewise, the work of P. Amer [12] deals with specifying a partial order based transport service for multimedia data. The service specification method is oriented towards developing validation models for temporal specifications of distributed multimedia applications. Such a transport service first needs to be incorporated into distributed programming primitives before use in developing applications.

As can be seen, our $Occurs\_After$ construct allows explicit specification of message causality, using an 'object-oriented programming' framework that transforms the processing of causally ordered messages into object-level 'state changes' occurring over specific real-time intervals.

6 Conclusions

The paper presented a model for synchronizing real-time data during presentation to the application. The basic premise in our approach is that the user level component of the communication system takes the burden of synchronization, instead of the network. This introduces flexibility in the message transport protocols and allows optimizing the usage of network resources to application needs.

The temporal properties of real-time presentation are specifiable in the form of a *causal ordering* on the data segments, i.e., messages, flowing across application entities. In this notion, application entities are modeled as objects, with the messages exchanged between various objects in a prescribed order and within prescribed intervals inducing 'state changes' in them. Since causal ordering of messages is amenable for easier implementation in a distributed system, our model may be viewed as generating a transport-oriented QOS description that is mappable into a specification on the required delay behavior of the underlying system. Such a mapping is more directly usable in an implementation of presentation protocols than the higher level petrinet based specifications typically employed.

Our model of causal ordering allows application characteristics to be mapped into a set of data delivery procedures composed in the form of data presentation protocol. A possible relaxation of the data delivery constraints and the network delay requirements, as allowed by the model, offers a potential for optimal usage of system resources. The specification method is itself independent of how the client and server modules are separated in applications, which allows easier construction of programming models of real-time applications for analysis and verification, and a uniform implementation of communication systems. Furthermore, our specification model can be recursively employed in a hierarchically decomposed real-time presentation system.

References

1. G. Blair, G. Coulson, M. Papathomas, P. Robin, J. S. F. Horn, and L. Hazard. A Programming Model and System Infrastructure for Real-time Synchronization in

Distributed Multimedia Systems. IEEE Journal on Selected Areas in Communications, vol.14, no.1, (1996), 249-263.
2. L. Lamport. Time, Clocks and Ordering of Events in Distributed Systems. Communications of the ACM, (1978).
3. R. D. Hill. Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction — The Sassafras UIMS. ACM Transactions on Graphics, vol.5, no.3, (1986), 179-210.
4. Worldwide Web Consortium. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. P. Hoschka (ed.), W3C Recommendation, (1998).
5. B. Field, T. F. Znati, and D. Mosse. V-NET: A versatile Network Architecture for Flexible Delay Guarantees in Real-time Networks. IEEE Transactions on Computers, vol.49, no.8, (2000), 841-858.
6. R. Steinmetz. Synchronization Properties in Multimedia Systems. IEEE Journal on Selected Areas in Communications, vol.SAC-8, no.3, (1990), 401-412.
7. K. Birman and T. A. Joseph. Exploiting Virtual Synchrony in Distributed Systems. 11-th Symp. on Operating System Principles, ACM SIGOPS, (1987).
8. L. L. Peterson, N. C. Buchholz and R. D. Schlichting. Preserving and Using Context Information in Interprocess Communication. ACM Transactions on Computer Systems, vol.7, no.3, (1989), 217-246.
9. K. Ravindran and A. Thenmozhi. Extraction of Logical Concurrency in Distributed Applications. Proc. Intl. Conf. on Distributed Computing Systems, IEEE-CS, Pittsburgh (PA), (1993).
10. R. Yavatkar. MCP: A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications. Proc. Intl. Conf. on Distributed Computing Systems, IEEE-CS, Yokohama (Japan), (1992), 606-613.
11. J. P. Courtiat, L. Carmo, and R. Oliviera. A General-Purpose Multimedia Synchronization Mechanism Based on Causal Relations. IEEE Journal on Selected Areas in Communications, vol.14, no.1, (1996), 185-195.
12. P. Amer, C. Chassot, T. J. Connally, M. Diaz and P. Conrad. Partial-order TRansport Service for Multimedia and Other Applications. IEEE/ACM Transactions on Networking, vol. 2, no. 5, (1994), 440-455.