, , 1–27 ()

Automating the Organization of Presentations for Playout Management in Multimedia Databases

VELI HAKKOYMAZ veli@ces.cwru.edu
GÜLTEKIN ÖZSOYOĞLU tekin@ces.cwru.edu
Department of Computer Engineering and Science, Case Western Reserve University, Cleveland, OH 44106 USA

Received May 1, 1991

Editor:

Abstract. We introduce a constraint-driven methodology for the automated assembly, organization and playout of presentations from multimedia databases. We use inclusion and exclusion constraints for extracting a semantically coherent set of multimedia segments. Presentation organization constraints are utilized for organizing the multimedia segments into a presentation, which in turn helps decide the playout order of the extracted multimedia segments. The playout order of the segments is represented in a presentation graph. If the specified set of organization constraints are not sufficient to construct a unique presentation graph, we propose two techniques so that a unique graph is constructible. We also propose two playout algorithms, one for the generation, start and termination of playout agents, the other for dynamic control of playout management on organized presentations. The characteristics of these algorithms are expressed in terms of presentation playout parameters.

Keywords: Multimedia Presentation, Presentation Organization, Playout Synchronization, Synchronized Presentation, Multimedia Databases

1. Introduction

In multimedia computing research, organizing various multimedia segments for a semantically coherent presentation, without any regard to the multimedia database that contains these segments, is an active research area under the title of *presentation managers*. Multimedia computing integrates different media types such as audio, video, text, and graphic images. Each medium can be modeled as a stream [12, 7, 6, 3, 18] which can be broken into a sequence of segments. A multimedia presentation refers to the presentation of multimedia segments using a number of output devices such as speakers for audio, monitors for text and video and so on. In this paper, we discuss the issues involved in automatically organizing presentations from multimedia databases. In particular, we introduce a methodology to automate the organization and playout of multimedia presentations.

The organization of presentations is a complex task in that the display order of presentation contents (in time and space) must be specified. Suppose that an education technologist is developing a presentation called *Training* that contains audio, video, and text media types. The critical decisions for presentation construction

include (1) what the contents are, and (2) how the contents are organized (i.e., some parts of audio and video may be temporally related and have to be presented in parallel; some other parts can only be presented after certain subjects are covered, etc). Once the decision is made on the organization of the contents of the presentation, it must be conveyed to the end user in the correct organizational order and in a timely fashion.

In this research, we consider an environment where users request multimedia presentations of a fixed time length. For example, a user may request a one-hour long audio-video summary of a presentation about the programming language C^{++}. We also envision that the user (in some way) can specify the maximum number of monitor windows which can be open simultaneously for the presentation of parallel streams. Such a process can have multiple interactions between a user and the presentation system. For example, the system may respond to a user request by saying that the requested presentation cannot be performed in one hour if it is to be presented using the specified number of parallel windows. To automate the construction and playout of presentations, we propose a "constraint-driven approach". More specifically, we utilize

- *inclusion and exclusion constraints* for extracting a semantically coherent set of multimedia segments from the multimedia database,
- *presentation organization constraints* for organizing and deciding the playout order of the extracted multimedia segments,
- *playout-control constraints* in order to provide dynamic, playout-time controls for end users, and
- *physical-playout constraints* for helping to ensure a jitter- or hiccup-free playout of multimedia data.

We use inclusion and exclusion constraints between segments to facilitate the automated inclusion or exclusion of segments into a presentation. Consider an educational math lecture in video. In any presentation that contains a video sequence *Proof* illustrating the proof of a theorem, another video sequence *Thm* that defines the theorem should also be included. This is an *inclusion requirement* of *Thm* based on the included segment *Proof*. However, it is clear that *Thm* can be included in a presentation without including *Proof*. To summarize, when a user specifies (by pointing and clicking) a set of segments for a presentation, the $DBMS$, by using inclusion and exclusion constraints, adds segments into and/or deletes segments from the set in order to satisfy the inclusion and exclusion constraints. In a recent work[18] we have characterized inclusion and exclusion dependencies, axiomatized a subset, given two algorithms for automated presentation assembly, and discussed their complexity.

Presentation organization constraints allow the system to automate the organization of concurrent presentations of selected segments (that already satisfy inclusion and exclusion constraints). We assume that presentation organization constraints

are entered into the database a priori by the database administrator, and, for any set of user-selected segments, the satisfaction of presentation organization constraints leads to an organized presentation. Consider the educational math video example. The video sequence *Proof* must be preceded (but not necessarily immediately) by another video sequence, say *Thm*, that defines the theorem. This is a *sequentializer constraint* for a presentation that contains *Thm* and *Proof* (as *Thm* should precede *Proof*). In this paper, we discuss how to organize a concurrent presentation (represented by a "presentation graph") by utilizing the presentation organization constraints. We assume that the database contains various presentation organization constraints. Users express a presentation organization query by specifying (a) an upper bound on the time length of the presentation, (b) an upper bound on the number of parallel monitor windows (for video playout) open at any time, (c) a set of selected segments (which are expanded, if necessary, into a set of "coherent" segments by utilizing inclusion and exclusion constraints and the algorithms of our earlier work [18]). Note that the requirement (b) specifies the maximum level of concurrency (i.e., the number of concurrently played-out video segments) at a given time. Since a computer monitor has a physical size limit, it (and, perhaps the computing power of the playout environment) has an upper bound on the number of concurrent segments (i.e., windows) it can effectively play. For example, clearly, playing out over 6 concurrent (monitor) windows at a given time is excessive. Such a requirement is captured by the requirement (b) above.

There is a tradeoff to satisfying requirements (a) and (b) at the same time, and we show in section 4 that this is an NP-complete problem. Therefore, rather than finding the optimum solution, we propose and evaluate two heuristics that allow us to obtain a unique presentation graph satisfying the requirements (a) and (b) in a near-optimum manner.

Once a concurrent presentation is specified (using the presentation graph), it needs to be played out. We associate a *playout agent* to each segment in the presentation, which is a lightweight process (a thread) that plays out the corresponding multimedia segment. In section 5.1, we describe a semaphore-based technique for the automated generation, synchronization and termination of playout agents in order to implement the concurrent presentation playout as defined by the presentation graph.

Another issue is to automate the incorporation of playout-time controls into the assembled presentation. Let A denote the action "Freeze all video streams that are currently being played out for 20 seconds". The presentation assembly system may designate a keyboard function key that, when pressed, sends the signal S which indicates to the "playout manager" software that the action A has to be taken. This is an example of the incorporation of event-action rules from active databases into the automated presentation organization problem. (One can extend this model to incorporate nonevents (negative events) and/or prohibited actions [13], as well.) When such rules, i.e., constraints, are incorporated into the automated presentation assembly problem, there is a tradeoff between their satisfaction and the satisfaction of other constraints. For example, when the above freeze occurs, the presentation

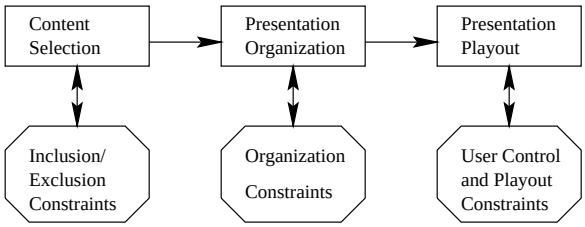

Figure 1. Multimedia Presentation System

time deadline may no longer be satisfiable, and a playout-time reorganization of the presentation graph may become necessary. We call such rule-based constraints *playout-control constraints*. In this paper, we discuss playout-control constraints with examples only.

Issues such as physical playout constraints, hiccup-free playout, quality of service (QoS) guarantees and so on, which are closely related to media presentation, are not the focus of this paper. The approach we have taken in addressing these issues is described in [10].

Figure 1 shows the constraint types and their functionality in our multimedia presentation system. In the rest of this section, we briefly survey the related work on presentation and playout management issues. Section 2 presents the basic definitions used throughout this paper. Section 3 characterizes the organization constraints. In section 4, we formally define the presentation organization problem and discuss two heuristic methods as an approximation to its solution. In section 5, after organizing a presentation in section 4, we present two methods for the presentation playout. Section 6 gives the concluding remarks and the direction of future research.

1.1. Related Work

In recent years, attempts to tackle the problem of preparing multimedia segments into a multimedia presentation and conveyance of the resulting presentation for human users have gained momentum in the literature[14, 11, 5, 12, 15, 6, 18, 24, 19].

Little and Ghafoor in [15] made one of the earliest attempts to develop a temporal-interval based(TIB) model that captures the timing relationships among multimedia data segments. They assume that inter-segment temporal relations are either imposed at the creation time of the multimedia segments (i.e., called live synchronization) or set up artificially (i.e., called synthetic synchronization). In their work, presentation of each multimedia data segment is represented by a time interval (start time, end time, duration). Using this model with the timing information, they come up with a playout schedule for the segments with 'monotonically increasing deadlines' in order to present them in a timely manner. To this extent, they

specify the temporal access algorithms to facilitate *forward* and *reverse playout* as well as *partial-interval evaluation* (for pause-resume operations).

The same problem is tackled by Hoepner [11, 12]. In these works, three distinct problems in multimedia presentations are identified as determining the contents as well as the layout of the presentation in time and space. However, the main focus of the work is concerned with the description of temporal aspects of an abstract presentation behavior. Synchronization and control of temporally related presentation actions are modeled by presentation frame types, *sequentializer, parallelizer, splitter, combiner, and brancher.*

These works mainly deal with modeling the way in which multimedia data segments are presented, but not the contents or the organization of multimedia presentations. As for the feasibility of providing presentations from multimedia databases over a distributed multimedia system, there have been several studies in describing the requirements and design of such systems[24, 19, 17].

Znati and Field in [24] focus on the design of the communication protocols, called ϕ-channel, to support guaranteed real-time communication for distributed multimedia systems. The ϕ-channel is a network level abstraction of a fractional, simplex, end-to-end communication channel between a source and a destination to support the requirements of real-time applications.

Another work in [19] by Qazi, Woo and Ghafoor states the need for a specification model for the communication and synchronization of multimedia segments in a distributed environment in order to realize successful retrieval, composition and presentation of multimedia segments.

2. Preliminary Definitions

We first identify the basic presentation organization constraints that enable users to express the flow of a multimedia presentation in terms of how and what order the segments are played out to the user, be it sequential, concurrent, or some combination of both. In this simple model, no time is involved in expressing the organization of multimedia segments. An expert user or presentation generator can express the presentation flow by specifying the presentation organization constraints for the multimedia segments that are contained in the presentation.

Basically, some segments may have to be played out in a sequential manner (i.e., one is before the other). We represent such a constraint between two segments (i.e., say a and b) as **sequential(a,b)** in the textual form and call it a **sequentializer** constraint.

Endings of the playout of some segments may signal the presentation to split into two or more streams, starting with a segment from each stream. Such a constraint involves at least three segments, one of which is considered to be a *designated segment.* After presenting the designated segment, the remaining segments are presented in parallel. We represent such a constraint among three segments (i.e., say a, b and c) as **split(a, b, c)** in the textual form, specify the first argument a as the designated segment, and call the constraint a **splitter** constraint.

The last presentation organization constraint type is the **merger** constraint, which indicates that, after two or more segments are presented in parallel (i.e., concurrently), they will merge into one stream, from which a designated segment will be presented. Like the splitter constraint, the merger constraint involves at least three segments, one of which is the designated segment. We represent such a constraint among three segments (i.e., say a, b, and c) as **merge(**a, b, c**)** in the textual form and specify the last argument c as the designated segment.

Note that no synchronization points (i.e., time values) are specified in this model. Only relative playout timings of segments are known.

A **presentation graph** $G = (V, E)$ is a directed graph which is augmented by two special nodes, *initial node* I and the *final node* F, where nodes in $V(G)$ are labeled with the segments in the presentation, and edges in $E(G)$ indicate the relative presentation order of two segments (i.e., $a \longrightarrow b$ specifies that segment a is "before" segment b in the presentation). Edges are added to a presentation graph according to the specified organization constraints. Figure 2(i) depicts a one-to-one correspondence between organization constraints in textual form and the corresponding graph components.

2.1. Obtaining Subpresentations

A subpresentation corresponds to a connected graph and we informally use the term **subpresentation** to refer to a structure that contains a collection of segments and organization constraints from which a presentation graph is constructible. A more descriptive definition will be given shortly.

Let $SS = (s_1, s_2, \cdots, s_n)$ denote the selected set of multimedia segments that are to appear in the presentation. Let $OC = (o_1, o_2, \cdots, o_t)$ denote the presentation organization constraints that are specified for SS.

Given any sets SS and OC, the subpresentations are constructed in two stages. In the first stage, the organizationally related segments are grouped together, and in the second, after augmenting with two extra nodes and a number of edges, each group becomes a subpresentation.

Stage 1: Grouping the Segments

1. Create a group C_i for each segment $s_i \in SS$.
2. Consider each organization constraint $o_i \in OC$ (in some arbitrary order).
   (A) If o_i is of type *sequential(*s_j *,* s_k *)*
      i. Let C_1 be the group that segment s_j belongs to, and let C_2 be the group that segment s_k belongs to.
      ii. Create a new group C_3 such that $C_3 = C_1 \cup C_2$.
      iii. Eliminate the groups C_1 and C_2 from further consideration.
   (B) If o_i is of type *split(a, S)* or *merge(S, a)* where $\mid S \mid = n \geq 2$,

i. Let C_{n+1} be the group that a belongs to, and let $C_1, C_2, \cdots, C_n$ be the n groups that n segments in S respectively belong to.
ii. Create a new group C_{n+2} such that $C_{n+2} = C_1 \cup C_2 \cup \cdots C_n \cup C_{n+1}$.
iii. Eliminate the groups $C_1, C_2, \cdots C_{n+1}$ from further consideration.

Let us assume that, after the segments are grouped, we end up with m groups, which we arbitrarily name as C_1, C_2, $\cdots$, C_m by changing the indices.

Stage 2: Augmenting Groups into Subpresentations
Consider each group C_i, and create a graph G_i with the following nodes and edges ($1 \leq i \leq m$):

1. Make a node for each segment in C_i and label it with the segment name.
2. For each organization constraint (as shown in Figure 2(i)) involving the segments in group C_i,
   (A) if it is of type *sequential(a, b)* where $a, b \in C_i$, add a directed edge from node a to node b (i.e., $a \longrightarrow b$) into the graph G_i.
   (B) if it is of type *split(a, S)* where $a \in C_i$ and $S \subseteq C_i$, add directed edges from node a to node s_i for each segment $s_i \in S$ (i.e., $a \longrightarrow s_i$) into the graph G_i.
   (C) if it is of type *merge(S, a)* where $a \in C_i$ and $S \subseteq C_i$, add directed edges from node s_i to node a for each $s_i \in S$ (i.e., $s_i \longrightarrow a$) into the graph G_i.
3. Determine the number of incoming and outgoing edges (i.e., *incoming_edge_count* and *outgoing_edge_count*, respectively) for each node (i.e., segment) in G_i.
4. For all nodes $v \in G_i$,
   (A) if $incoming\_edge\_count[v] = 0$ then add a directed edge from *the initial node* I to node v (i.e., $I \longrightarrow v$) into the graph G_i.
   (B) if $outgoing\_edge\_count[v] = 0$ then add a directed edge from node v to the *final node* F (i.e., $v \longrightarrow F$) into the graph G_i.

Nodes I and F represent the start and terminate nodes, respectively, for each subpresentation. They both are empty (null) segments. After the augmentation, we call each graph G_i a *subpresentation*.

Example: Using the above procedure, we form a unique subpresentation graph from a given set of segments and organization constraints. Figure 2 depicts the way the construction algorithm works. □

A multimedia presentation can be described as a particular arrangement for a collection of subpresentations. Each subpresentation G_i has a *source* (i.e., the segment that is to be presented first which is I in our case) and a *sink* (i.e., the segment that is to be presented last which is F in our case). Every node in a subpresentation is related to every other node through some specific temporal relation.

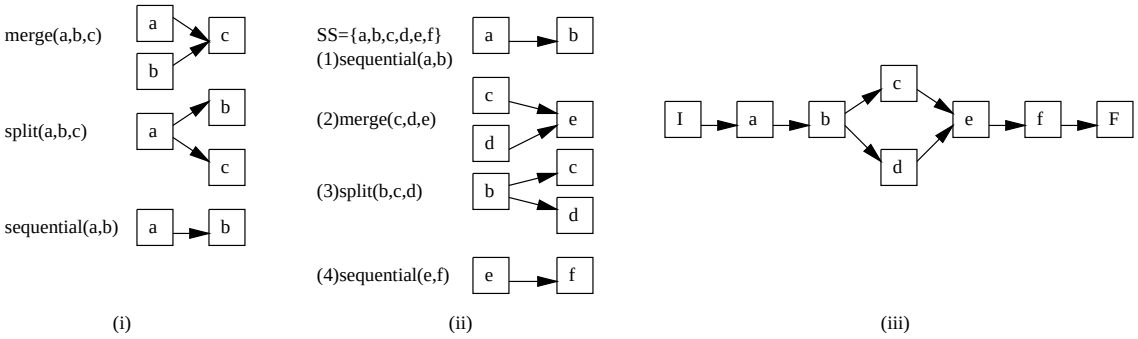

Figure 2. Relationship between Organization Constraints and Presentation Graph. (i)Simple organization constraints and their transformations into graph components, (ii) a set of constraints and the corresponding graph components, (iii) resulting presentation graph.

Each subpresentation has a *length* (i.e., the maximum of the sums of the lengths of all segments on each path from the source to its sink) as well as a *height* (i.e., the maximum cut[16]) that is computed in a way that will be described shortly. We will also use the notion of *height at a point* for a subpresentation. Notice that I and F nodes are introduced in each subpresentation. To provide unique node labels for a presentation graph constructed out of several subpresentations, we may easily rename I and F nodes of each subpresentation with unique labels (i.e., new indices). Yet these are the empty (null) nodes in the presentation graph. Thus, each presentation graph can be made to have only one unique I (initial) node and one unique F (final) node by introducing new indices.

Let G_i denote such a subpresentation for a presentation consisting of n subpresentations, $1 \leq i \leq n$. A particular arrangement for a collection of subpresentations means that all G_i's are merged into a single connected (presentation) graph G in such a way that (1) the user-specified limit on the presentation length is not exceeded, (2) the height of the resulting presentation is less than a user-specified height limit. As an illustration, the maximum number of video segments that are presented in parallel must not exceed the available number of monitor windows for video (specified by the user). We use the term *height* to refer to the available number of monitor windows for video. In other words, "joining" a set of G_i's into a single directed graph means identifying the presentation organization constraints among all the G_i's.

3. Presentation Organization

In this section we characterize presentation organization constraints and discuss the presentation organization issues.

3.1. Presentation Organization Constraints

The purpose of presentation organization constraints is to automate the organization of a concurrent presentation containing the selected set SS of multimedia segments (which already satisfy presentation inclusion and exclusion constraints). We assume that presentation organization constraints are entered into the database a priori by the database administrator.

To summarize, we use the following presentation organization constraints:
Sequentializer (SQ) constraint between segments a and b: In any presentation with a and b, presentation of a is succeeded by the presentation of b.
Splitter (SP) constraint between segment a and segment set S: When a and any subset S' of S is in a presentation, the presentation of a is succeeded by a concurrent presentation of all the segments of S'.
Merger (MG) constraint between segment a and segment set S: When a and any subset S' of S are in a presentation, after all segments in S' are concurrently presented, they are merged into a single stream, and a is presented.

3.2. Interpreting Presentation Organization Constraints

Sequential(a, b) is equivalent to *"a meets b"* [1] where a and b represent two time intervals, and their relationship is that the interval b starts at the point where interval a ends. Assuming that S is a nonempty set of segments, two other organization constraints can be interpreted as follows: *Split(a, S)* is equivalent to *"a meets s_i"* $\forall s_i \in S$.

Consider the merger constraint *merge(S, a)*. In one case, segment a can start right after at least one of its immediate predecessors ends (i.e., *at-least-one semantics*). We can, therefore, set the starting time of the segment a as the ending time of the earliest ending predecessor segment. In another case, segment a can start only after all of its immediate predecessors end (i.e., *all semantics*). For this semantics, we have to set the starting time of the segment a as the maximum of the ending times of all the segments that immediately precede the segment a.

For the merger constraint, we can express these two semantics as follows: Let a be a segment, ST be the segment start time, and l be the playout duration of the segment. Then, for all segments p, where p is an immediate predecessor of segment a,

All Semantics:

$$ST(a) = \begin{cases} 0 & \text{if } a \text{ is a source} \\ max\{ST(p) + l(p)\} & \text{otherwise} \end{cases} \quad (1)$$

At-least-one Semantics:

$$ST(a) = \begin{cases} 0 & \text{if } a \text{ is a source} \\ min\{ST(p) + l(p)\} & \text{otherwise} \end{cases} \quad (2)$$

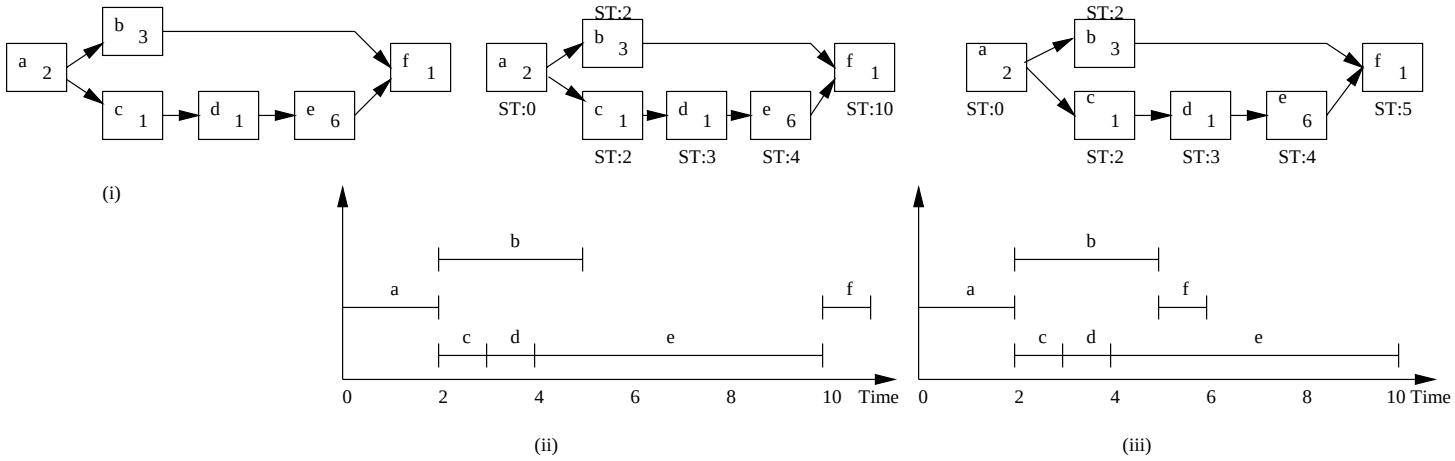

Figure 3. Computation of Segment Start Times Using Two Semantics of Organization Constraints. (i) a presentation graph, (ii) The same presentation graph with segment start times(ST) using the all semantics, (iii)The same presentation graph with segment start times(ST) using the at-least-one semantics

Example: Using the semantics defined above, we can compute the start times (ST) of segments in the presentation graph as shown in Figure 3(i). The playout duration l of each node's segment is indicated in the lower-right corner of the corresponding node.

Using the all semantics, the upper graph in Figure 3(ii) shows the presentation graph of Figure 3(i) with the computed start times for the playout of each segment. The lower part of Figure 3(ii) shows the playout times of segments using a timeline diagram.

Using the at-least-one semantics, the upper graph in Figure 3(iii) shows the presentation graph of Figure 3(i) with the computed start time for the playout of each segment. The lower diagram of Figure 3(iii) shows the playout times of segments using a timeline diagram. □

Note that there exists an anomaly for at-least-one semantics in the timeline diagram of Figure 3(iii): Even though the node f is designated as the final node in the presentation graph, it ends much earlier than one of its predecessors, namely, the segment e. Because of this anomaly, we use the all-semantics for the merger constraint in our examples in the rest of the paper.

3.3. Controlling the Length and Height of a Presentation

We call the set OC of presentation organization constraints a **cover** if a unique presentation is obtained by enforcing the constraints in the set OC. Assume that

a set of presentation organization constraints is a *cover* for a given collection of multimedia segments. In this case, the task of forming a presentation is straightforward. After forming a presentation graph as described earlier, we use it to find the longest path from the initial node I to the final node F. The longest path gives us the length of the presentation. To satisfy the user-specified presentation length limit, the only thing that needs to be done is to check whether or not the user-specified length exceeds the computed presentation length.

As for the height computation, a simple algorithm that is presented in the next subsection can be used on the presentation graph. This algorithm finds the number of streams that need to be presented in parallel. As the number of parallel streams played out vary with time, we are only interested in the maximum number. By applying the algorithm between the start and end points of the presentation, we obtain a set of numbers, the maximum of which gives us the height of the presentation (i.e., the maximum number of monitor windows required to play out this presentation). We then check whether or not the user-specified height exceeds the computed presentation height.

However, the assumption of having presentation organization constraints forming a cover, and thus, leading to a unique presentation graph usually does not hold for real-world applications. In this case, there is at least one segment a that can be played out in parallel or sequentially with some other segment c without violating any presentation organization constraint in the set.

3.4. Computing the Height of a Presentation

Given a subpresentation with a presentation graph, what is the maximum number of parallel windows needed for playing the video segments in the subpresentation? This problem is identical to finding the maximum "cut" in a given temporally-aligned graph. Therefore, the maximum cut corresponds to the height of the presentation. We give the following simple algorithm for this purpose.

Call the start and the end of each segment in a subpresentation as the *start* and *end event* or simply *"event"*. Assuming that there are n segments in a subpresentation, the total number of events is $2n$. Events occur at specific time points which are called *event points*. The total number of event points can be fewer than the total number of events since multiple events may occur at the same event point. Therefore, the total number of event points is at most $2n$.

Since event points are time values (i.e., positive numbers), they can be ordered in increasing/decreasing order. Let m denote the total number of event points, where $m \leq 2n$, and $Event\ i$ denote the group of events that occur at the i^{th} event point, $1 \leq i \leq m$.

Height at a point, height of a subpresentation at any point in time (i.e., say, at time d) equals the height between event points for $Event\ j$ and $Event\ j+1$ where (1) event point for $Event\ j \leq d$, and (2) there does not exist an $Event\ i$ such that event point for $Event\ j <$ event point for $Event\ i \leq d$, and (3) $j < m$. This is true

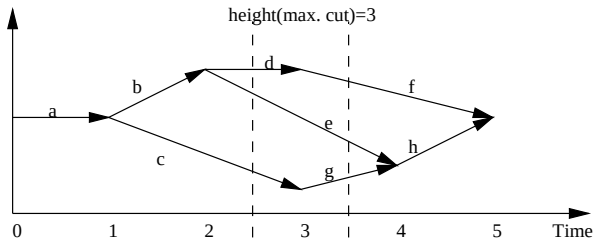

Figure 4. A Timeline Diagram Showing the Event Points of a Subpresentation Graph

because no new edge appears in the presentation graph between two consecutive event points.

Formally, an *Event* will be characterized by $[E, S, T]$ where E is the event point (i.e., a positive number), S is the set of segments that start at E, T is the set of segments that end at E. After ordering the events in increasing event point order, we denote each event with an index $[E, S, T]_i$ for the i^{th} event, $i \leq m$, or an element of an event i as E_i, S_i, or T_i. For a subpresentation graph, since all the segments that start will eventually end, the following holds:

$$\sum_{i=1}^{m} \mid S_i \mid = \sum_{i=1}^{m} \mid T_i \mid \tag{3}$$

The height between any two consecutive event points (E_k and E_{k+1}, $k < m$) for a subpresentation is expressed by :

$$\sum_{i=1}^{k} \mid S_i \mid - \sum_{i=1}^{k} \mid T_i \mid \tag{4}$$

Example: Figure 4 depicts the segment playout behavior using a timeline diagram for a subpresentation graph. Using the notations introduced so far, the following events characterize this subpresentation (n=8; m=6):

Event 1 : [0, {a}, {}] *Event* 4 : [3, {g,f}, {c,d}]
Event 2 : [1, {b,c}, {a}] *Event* 5 : [4, {h}, {g,e}]
Event 3 : [2, {d,e}, {b}] *Event* 6 : [5, {}, {h,f}]

The heights between two consecutive event points for this subpresentation graph are computed as follows:

interval $[0, 1) = 1 - 0 = 1$; interval $[3, 4) = 7 - 4 = 3$;
interval $[1, 2) = 3 - 1 = 2$; interval $[4, 5) = 8 - 6 = 2$;
interval $[2, 3) = 5 - 2 = 3$; interval $[5, \infty) = 8 - 8 = 0$;

Thus, the height of this subpresentation is 3. □

4. Problem Definition

Recall that the subpresentations are $G_1, G_2, \cdots, G_n$.

First, let us define what we mean by the term **arrangement**: An arrangement of subpresentations means that each subpresentation G_j is connected to some other subpresentation(s) G_i and G_k without forming a cyclic graph in such a way that (1) the source node of G_j is an immediate successor of the sink node of G_i or the initial node I, (2) the sink node of G_j is an immediate predecessor of the source node of G_k or the final node F. Recall that I and F nodes in subpresentations can be renamed (with new indices) so that nodes in the resulting presentation graph has unique labels.

Definition. Presentation Organization Problem (POP):
Instance : Given (1) a set P of n subpresentations $(G_1, G_2, \cdots, G_n)$ each of which with a height and a length denoted by $h(G_i)$ and $l(G_i)$ for $1 \leq i \leq n$, respectively, (2) a positive number, $Length \geq 0$, and (3) a positive integer, $Height \geq 0$,
Question : Does there exist an arrangement M for P such that (1) $length(M) \leq Length$, and (2) $height(M) \leq Height$?

We will consider a simplified version of the POP problem, called S_POP, where all the subpresentations are of equal length (i.e., $l(G_i) = c, \forall G_i \in P$ and c is a constant).

To determine the complexity of S_POP problem, we consider the following known NP-complete problem *Bin Packing* and find a polynomial time reduction from *Bin Packing* to S_POP.

Definition. Bin Packing Problem:
Instance : Given (1) a finite set U of items, a size $s(u) \in Z^+$ for each $u \in U$, (2) a positive integer bin capacity B, and (3) a positive integer K,
Question : Is there a partition of U into disjoint sets $U_1, U_2, \cdots, U_K$ such that the sum of the sizes of the items in each U_i is B or less?

THEOREM 1 S_POP *is NP-hard.*

Proof: We give the following transformation (reduction) from Bin Packing to S_POP:

The set U of items in Bin Packing is the same as the set P of subpresentations in S_POP. Each item has a size $s(u) \forall u \in U$ in Bin Packing, while in S_POP each subpresentation has a height $h(G_i), \forall G_i \in P$. The bin capacity B of Bin Packing corresponds to the user-specified $Height$ in S_POP. The number of bins, K, transforms to the number of intervals, N, each of which has a length 'c'. Overall, the total length of all the intervals is determined as cN, which is the user-specified $Length$ in S_POP.

Now the question in Bin Packing becomes as:

Is there a partition of P into disjoint intervals $I_1, I_2, \cdots, I_N$, (each with a length c, thus the overall length cN) such that the sum of the heights of the subpresentations in each interval I_i is less than or equal to Height?

A "yes" answer to Bin Packing requires a "yes" answer to S_POP, and a "no" answer to Bin Packing requires a "no" answer to S_POP. It is obvious that the given transformation is polynomial. ■

As a consequence of the theorem, we no longer focus on finding an optimal solution to the POP problem, but instead attempt to find a "good" heuristic solution. The methods used for designing such algorithms tend to be problem specific [4, 8]. Two heuristics based on a combination of empirical studies and common-sense arguments are given next.

4.1. Heuristic 1 : Maximum Parallelism

This heuristic attempts to find the shortest-length presentation that satisfies all the organization constraints (i.e., note that subpresentations are constructed to satisfy the given set of presentation organization constraints), the user-specified limits on the presentation length and presentation height.

The length of a presentation is at least the maximum of the lengths of its subpresentations because, in one extreme case, all the subpresentations will be played out in parallel and the subpresentation with the maximum length will determine the length of the presentation. However, a parallel playout of all the subpresentations may not satisfy the user-specified height limit. Therefore, some of the subpresentations may need to be organized back-to-back (i.e., sequential). Heuristic1 first sets the presentation height with the user-specified *UHeight* and then tries to form a presentation with the minimum possible length still satisfying the POP problem. To do so, a subset of subpresentations are selected and linked together without exceeding the user-specified height and length limits. If the length limit is not exceeded and all the subpresentations are linked together, then the algorithm finds a presentation and declares "success". However, if there are subpresentations which are not linked yet, and the length limit *ULength* is not exceeded by the current length, then the length is incremented by a certain amount and the same process is repeated. In case the length is incremented above a threshold value, *ULength*, then the algorithm cannot find a presentation and declares "failure". The pseudo code of the Heuristic1 is shown below:

```
1.  Algorithm Heuristic1(UHeight, ULength, P):
2.  Input:
3.      UHeight, ULength: User-specified Height and Length bounds
4.      P: set of subpresentations G1, G2, ..., Gn
5.      each of which is given with h(Gi) and l(Gi)
6.  Output:
7.      success/failure in forming a presentation
8.  Body:
9.      X := Policy1;
10.     TLength := 0;                          /* tentative length */
11.     Remain := P;                           /* set of unrelated Gi's */
12.     Pred := {I}; Presentation := {I};
```

```
13.  /* I is the initial, F the final node */
14.       while Remain is nonempty do begin
15.            Choose (wrt X) a subset R of Remain such that
16.                 ∑_{G_i ∈ R} h(G_i) ≤ UHeight;
17.            TLength := TLength + max{l(G_i), ∀G_i ∈ R};
18.            if TLength>ULength then begin
19.                 TLength := 0; Remain := P;
20.                 Pred := {I}; Presentation := {I};
21.                 switch(X)
22.                 case Policy1: X := Policy2;
23.                 case Policy2: X := Policy3;
24.                 case Policy3: Exit with failure;
25.            end
26.            else begin
27.                 Remain := Remain - R;
28.                 Add R into Presentation by making every subpresentation in
29.                 Pred(ie, most recently added subpresentations) an immediate
30.                 predecessor of each subpresentation in R;
31.                 Pred := R;
32.            end
33.       endwhile
34.       Add {F} into Presentation by making it an immediate successor of every
35.       G_i in Pred;
36.       Return success with Presentation;
```

At each iteration within the while loop, choosing a subset of subpresentations can be performed in many different ways. Currently, we have three policies:

- Policy1 : Choose the subpresentations with "small" heights, the summation of which must not exceed the prespecified height.

- Policy2 : First choose subpresentations with "large" heights, the summation of which must not exceed the prespecified height. Then choose subpresentations with "small" heights, where the total height is less than Uheight.It is possible that we can still fit more subpresentations with small heights into the presentation without exceeding the prespecified height. Therefore, in this policy, we add into the presentation subpresentations with large heights first, and then subpresentations with small heights.

- Policy3 : Choose subpresentations in random.

4.2. Heuristic 2 : Steady Flow

This heuristic attempts to find the lowest-height presentation that satisfies all the organization constraints, the user-specified limits on the presentation length and the presentation height.

The height of a presentation is at least the maximum of the heights of its subpresentations because, in one extreme case, all of the subpresentations will be played out sequentially and the subpresentation with the maximum height will determine the height of the presentation. However, a sequential playout of subpresentations may not satisfy the user-specified length limit. Therefore, Heuristic2 first tries to form a presentation with the minimum possible height still satisfying the POP problem. To do so, it sets the tentative presentation height with the maximum of the heights of subpresentations. Afterwards, the subpresentations are selected and linked together incrementally without exceeding the user-specified height or length limits. If the length limit is not exceeded and all the subpresentations are linked together, then the algorithm succeeds in finding a presentation and declares "success". However, if the length limit is exceeded, then the tentative height is incremented by a certain amount, *delta*, and the same process is repeated. In case the tentative height is incremented above a threshold value, *UHeight*, then the algorithm cannot find a presentation and declares "failure". Note that this heuristic does not exclude the parallel arrangement of subpresentations. Therefore, some of the subpresentations may need to be organized in parallel. The pseudo code of the Heuristic2 is given below:

1. **Algorithm Heuristic2(UHeight,ULength,P,delta)**:
2. Input:
3. delta: the amount of increment
4. UHeight, ULength: User-Specified Height and Length bounds
5. P: set of subpresentations $G_1, G_2, \cdots, G_n$,
6. each of which is given with $h(G_i)$ and $l(G_i)$
7. Output:
8. success/failure in forming a presentation
9. Body:
10. X := Policy1;
11. THeight:= $\max\{h(G_i), \forall G_i \in P\}$; TLength := 0;
12. /* tentative height and tentative length */
13. Remain := P; /* set of unrelated Gi's */
14. Presentation := {I}; Pred := {I};
15. /* I is the initial, F the final node */
16. *while* Remain is nonempty *do begin*
17. Choose (wrt X) a subset R of Remain such that
18. $\sum_{G_i \in R} h(G_i) \leq THeight$;

```
19.          TLength := TLength + max{l(G_i), ∀G_i ∈ R}
20.          if (TLength > ULength) then
21.               if (THeight ≥ UHeight) then begin
22.                    THeight := max{h(G_i), ∀G_i ∈ P};
23.                    TLength := 0; Remain := P;
24.                    Pred := {I}; Presentation := {I};
25.                    switch(X)
26.                    Policy1: X:= Policy2;
27.                    Policy2: X:= Policy3;
28.                    Policy3: Exit with failure;
29.               end
30.               else begin
31.                    THeight:= min{UHeight, THeight+delta};
32.                    TLength:= 0; Remain:= P;
33.                    Presentation:= {I}; Pred:= {I};
34.               end
35.          else begin
36.               Remain := Remain - R;
37.               Add R into Presentation by making every subpresentation in
38.               Pred(ie, most recently added subpresentations) an immediate
39.               predecessor of each subpresentation in R;
40.               Pred := R;
41.          end
42.     endwhile
43.     Add {F} into Presentation by making it an immediate successor of every
44.     G_i in Pred;
45.     Return success with Presentation;
```

4.3. Empirical Evaluation of Heuristic1 and Heuristic2

What does the Implementation do: User specifies
(1) the desired presentation height, UHeight,
(2) the desired presentation length, ULength,
(3) number N of subpresentations to organize.

After these specifications are given, we do the following (in certain number of times.)
(i) a new seed value is determined for the random-number generator,
(ii) using the random-number generator, the N subpresentations are generated.

–the heights of the subpresentations are obtained from a uniform distribution in the range of (1..6).

–the lengths of the subpresentations are obtained from a uniform distribution in the range of (1..10).
(iii) subpresentations are ordered with respect to their heights. Now the Algorithms Heuristic1 and Heuristic2 can be applied on these subpresentations.

How the Heuristics work:

Heuristic1 :
For specified height,
(1) tries to find a presentation using policy 1,
If it succeeds, then exits with the presentation, otherwise
(2) tries to find a presentation using policy 2,
If it succeeds, then exits with the presentation, otherwise
(3) tries to find a presentation using policy 3, If it succeeds, then exits with the presentation, otherwise declares "failure".

Heuristic2 :
Same as the Heuristic1 except for the fact that Heuristic2 tries to find a "steady-flow" presentation.

What can we measure: To better understand the working principles of two heuristics and to see the effects of the user-defined parameters on each of them, we have performed a number of tests by simulating these two algorithms. In our simulation environment, the selected set of segments[1] for presentation are assumed to be grouped and augmented to 20 subpresentations according to the method described in section 2.1. Heights and lengths of subpresentations are obtained from a uniform distribution between (1..6) and (1..10), respectively. For each set of input parameters supplied by the user, we generate 1000 sets of these subpresentations and, for each set, try to form a presentation graph conforming to the user-constraints.

5. Presentation Playout

After a presentation is assembled which satisfies all the given presentation organization constraints, user-specified length and height limits on the presentation height and length, we now discuss dynamic playout control issues in an automated multimedia presentation environment.

Table 1. Number of times Heuristic1 forms a presentation(out of 1000 sets).

hts:len	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115
6	0	0	0	0	8	24	58	132	261	413	576	724	855	929	973	991	998	999
7	0	0	0	0	19	53	126	278	453	636	785	889	949	980	993	998	999	All
8	0	0	3	15	52	136	315	517	709	869	937	972	993	998	All	All	All	All
9	0	0	6	46	140	334	540	750	894	963	989	996	All	All	All	All	All	All
10	0	2	28	107	286	543	766	908	982	995	998	All	All	All	All	All	All	All
11	0	14	64	192	463	737	904	980	998	999	All	All	All	All	All	All	All	All
12	3	35	138	382	671	894	978	997	All	All	All	All	All	All	All	All	All	All
13	7	62	220	528	802	953	994	998	All	All	All	All	All	All	All	All	All	All
14	20	102	353	683	913	980	998	999	All	All	All	All	All	All	All	All	All	All
15	38	179	504	828	966	996	All	All	All	All	All	All	All	All	All	All	All	All
16	68	286	651	919	983	999	All	All	All	All	All	All	All	All	All	All	All	All
17	114	430	765	963	996	999	All	All	All	All	All	All	All	All	All	All	All	All
18	165	555	867	982	All	All	All	All	All	All	All	All	All	All	All	All	All	All
19	229	673	933	992	All	All	All	All	All	All	All	All	All	All	All	All	All	All
20	319	768	973	995	All	All	All	All	All	All	All	All	All	All	All	All	All	All
21	429	846	989	997	All	All	All	All	All	All	All	All	All	All	All	All	All	All
22	547	886	997	999	All	All	All	All	All	All	All	All	All	All	All	All	All	All
23	675	922	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
24	770	957	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
25	866	975	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
26	922	991	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
27	957	995	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
28	986	997	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
29	992	999	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
30	999	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All

5.1. Generation, Start, and Termination of Playout Agents

In order to describe the design of generic policies for a multimedia playout system in terms of timing parameters (of a real-time system [2, 9, 20], we first discuss our playout model and its characteristics.

We associate a *playout agent*(PA) to each segment to be presented. A PA is a schedulable entity (i.e., a lightweight process) within the operating system and responsible for playing out the associated segment. Segments are played out at appropriate times on specific types of output devices depending on the segment's media type. The following parameters describe the characteristics of a PA:

- *An arrival time* (denoted by a_i for PA segment i), at which the corresponding segment has been fetched from its source and brought to the presentation playout site.
- *An earliest start time* (denoted by r_i for PA for segment i), the PA can start playing out segment i anytime after r_i.
- *An actual start time* (denoted by p_i for PA for segment i) at which the PA starts playing out the segment i.
- *A playout duration* (denoted by x_i for PA for segment i), during which the PA continuously playout the segment i.
- *A deadline* (denoted by d_i for PA for segment i), by which the PA is expected to complete its playout of the segment.
- *An actual finish time* (denoted by f_i for PA for segment i), at which the PA completes playing out the segment i.

Using the notation introduced in section 3.4, we give, in Table 2, our first playout algorithm.

A PA associated with any segment a has the following pseudo code:
WaitforSV(a_{ST});
  Present segment a on its output device in time x_a;
SignalSV(a_T);

WaitforSV and SignalSV are binary semaphore primitives for synchronization purposes[17, 21, 22]. Concurrent activities typically require synchronization points so that the slow activities can catch up the faster ones, or the fast activities are slowed down to let the other not-so-fast activities to come close and meet them at these points. Each synchronization point is represented as a synchronization variable or SV (i.e., a binary semaphore). Such a variable, if set, indicates that the synchronization point has been reached, if unset, then this point has not been reached. For each segment a, we create two unique SVs, one for the start (a_{ST}) and one for the terminate (a_T) both are unset, initially. The following primitive

Table 2. First Playout Algorithm.

1. Order the events in the presentation in increasing order in event points.
2. Consider each event (in order) with event point E_i
   (A) For each segment v in T_i,
   Terminate the PA for v;
   SignalSV(v_T);
   (B) For each segment u in S_i,
   SignalSV(u_{ST});
   Start the PA for u;
   (C) Wait until the next event point E_{i+1}.

Table 3. Specification of first playout algorithm in terms of timing parameters.

Parm. setting	Meaning
$a_i = 0$;	All the segments are ready at the beginning at presentation playout site.
$p_i = r_i$;	All the PAs start playing out their segments at the earliest start time.
$d_i = r_i + x_i$;	All PAs must complete playing out their segments in x_i units of time.
$f_i = d_i$;	All the PAs complete playing out their segments at specified deadline.

operations are used on these variables: WaitforSV(SVExpr): to wait for the expression to become true. SignalSV(SV): to set the synchronization variable SV.
SV is a single synchronization variable and $SVExpr$ is an expression involving any number of SVs separated by *and* operator. At the beginning, all SVs are unset. A presentation playout session starts with SignalSV(I_{ST}). Using the parameters describing the characteristics of PAs, this playout algorithm can be expressed as in Table 3.

Notice that this algorithm is static, meaning that the behavior is predetermined. However, due to the media-related processing (for example, fetching, uncompressing or decoding very large video (MPEG) or audio files), or the heavy workload, CPU time is consumed. As a result, there will be delays in scheduling the PAs. The net effect is (1) information loss by skipping some frames, or (2) drop in frame rate, etc. To prevent such cases, we propose, in Table 4, a dynamic, on-line playout algorithm:

Example: Using the subpresentation graph shown in Figure 5, we can easily modify the PAs so that a complete synchronization of the playout activities is achieved.

Table 4. Second Playout Algorithm.

1. For each node v in the presentation graph, create two synchronization variables, one for start(v_{ST}) and one for terminate(v_T).
2. Generate a PA with the following (pseudo)code for each segment a:

   WaitforSV(a_{ST});

   Present segment a on its output device;

   SignalSV(a_T);
3. For each organization constraint (i.e, observed in the presentation graph),

   (A) if it is Sequential(a,b), add SignalSV(b_{ST}) at the end of PA for a.

   (B) if it is Split(a,b,c), add SignalSV(b_{ST}) and SignalSV(c_{ST}) (in this order) at the end of PA for a.

   (C) if it is Merge(b,c,d), add WaitforSV(b_T and c_T) and SignalSV(d_{ST}) (in this order) at the beginning of PA for d.

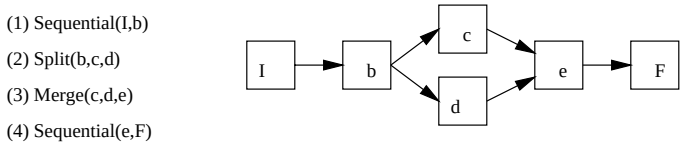

Figure 5. A Subpresentation Graph constructed out of constraints (1) to (4)

Table 5. Specification of second playout algorithm in terms of timing parameters.

Parm. setting	Meaning
$a_i = 0$	All the segments are ready at the beginning at presentation playout site.
$p_i = r_i \mp D_{i1}$	All PAs start playing out their segments in close range of their release times.
$d_i = r_i + x_i \pm D_{i2}$	All PAs can complete playing out their segments in the range of time x_i
$f_i = d_i \pm D_{i3}$	All PAs complete playing out their segments in the range of their deadlines.

PA for I:
WaitforSV(I_{ST});
present I;
SignalSV(I_T);
SignalSV(b_{ST});by (1)

PA for b:
WaitforSV(b_{ST});
present b;
SignalSV(b_T);
SignalSV(c_{ST});by (2)
SignalSV(d_{ST});by (2)

PA for c:
WaitforSV(c_{ST});
present c;
SignalSV(c_T);

PA for d:
WaitforSV(d_{ST});
present d;
SignalSV(d_T);

PA for e:
WaitforSV(c_T *and* d_T);by (3)
SignalSV(e_{ST});by (3)
WaitforSV(e_{ST});
present e;
SignalSV(e_T);
SignalSV(F_{ST});by (4)

PA for F:
WaitforSV(F_{ST});
present F;
SignalSV(F_T);

□

A presentation playout session starts with SignalSV(I_{ST}). In terms of characteristic timing parameters, this algorithm can be expressed as in Table 5.

By looking at the values of D_{i1}, D_{i2} and D_{i3}, one can study/observe the system characteristics in terms of where and how much delays/speedups occur in a presentation playout.

5.2. Execution-Time User Control During the Playout

To allow interactive user controls during the playout of an automatically generated presentation, an arbitrary number of (keyboard) buttons are reserved for the user interaction. Through these buttons, the user is able to change/affect the flow of the presentation in certain ways. The following examples illustrate such a control mechanism.

Example: 1. Pressing the function key $F1$ and entering the value y has the effect that all streams are frozen for y seconds.

2. Pressing the function key $F2$ and entering the value y has the effect that the specified streams are frozen for y seconds.

3. Pressing the function key $F3$ and entering the value x has the effect that, for the rest of the presentation, the maximum number of concurrent windows must be equal to x.

4. Pressing the function key $F4$ and entering the value x has the effect that, for the rest of the presentation, the minimum number of concurrent windows must be equal to x.

5. Pressing the function key $F5$ and entering the value y and object id of the content-object o has the effect that any stream containing a representative frame with content-object o is frozen for y seconds.

6. Pressing the function key $F6$ has the effect that all streams that are currently being played out are frozen until the *ContinueButton* is pressed.

7. Pressing the function key $F7$ and pointing and clicking a number of streams has the effect that the specified streams are frozen until the *ContinueButton* is pressed.

8. Pressing the function key $F8$ and entering the object id of the content-object o has the effect that any stream containing a representative frame with the content-object o is frozen until the *ContinueButton* is pressed.

□

All of these user controls can be modeled with the event-action paradigm of active databases[13, 20]. In general, $e \Longrightarrow A$ indicates that whenever the event e occurs, the action A must be taken. In all of the above examples, pressing a keyboard function key corresponds to an event, and its requested effect corresponds to the action taken by the presentation manager.

$Event_a$: Press function key $F1$ and enter value y.
$Action_a$: Freeze all streams for y seconds.
$Event_b$: Press the function key $F6$.
$Action_b$: Freeze all streams until the *ContinueButton* is pressed.

The overall meaning of these two events and actions are the same for our model: upon the occurrence of either of these two events, the presentation pauses for d time units, where d equals either y or the time duration between the occurrence of the event and the pressing of the *ContinueButton*. Figure 6 shows the significant points on a timeline during a presentation playout. Significant points are described below:

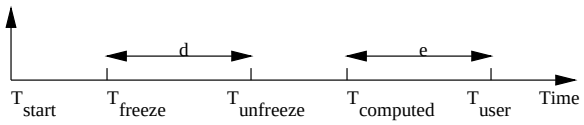

Figure 6. Significant time points during a presentation playout

T_{start} is the time point at which the playout of the presentation starts. T_{freeze} is the time point at which $Event_a$ or $Event_b$ occurs. $T_{unfreeze}$ is the time point at which the playout of the presentation resumes. $T_{computed}$ is the time point at which the playout of the presentation is supposed to end. T_{user} is the time point corresponding to the user-specified presentation length, $ULength$.

To analyze the incorporation of such event-action rules into our automated presentation organization model, we classify the segments of a multimedia presentation during a presentation playout into three groups: (i) Group D of segments that are already presented (i.e., Done), (ii) Group C of segments that are currently playing (i.e., Currently playing), (iii) Group Y of segments that are yet to play (i.e., Yet to play). For their incorporation, we create a new initial node I and make it an immediate predecessor of each segment in C (i.e. node that is labeled with the segment name) and the playout resumes with the newly added node I.

To determine whether we need a reorganization of the remaining portion of the presentation, we first compute the quantities d (i.e., $T_{unfreeze}$-T_{freeze}) and e (i.e., T_{user}-$T_{computed}$). Moreover, we find out the value of $rollback$=max{length(m_i), $\forall m_i \in C$}, since we may have to "rollback" the playout at most $rollback$ units to go back to the beginning of the segments that were playing at the time of the occurrence of the freeze event. The overall delay caused by this event is at most $d + rollback$. Therefore, if $d + rollback \leq e$, then no reorganization is needed. Otherwise, one has to do an incremental reorganization. This issue is currently being researched.

6. Conclusion and Future Work

In this paper, we have introduced a constraint-driven methodology for the automated assembly, organization and playout of presentations from multimedia databases. It is shown that the presentation organization problem is nontrivial. If the specified set of organization constraints are not sufficient to construct a unique presentation graph, we propose heuristic techniques so that a unique graph is constructible. After a presentation graph is constructed, we propose two playout management techniques, one for the generation, start and termination of playout agents, the other to provide dynamic (playout-time) controls for playout management. The playout characteristics are expressed in terms of presentation playout parameters.

We are currently implementing a prototype system to observe/understand the playout characteristics by experimenting with the timing parameters, which we in-

tend to use in a multimedia presentation system. For the time being, a single host site is responsible for (1) the selection of presentation contents, (2) organizing the selected contents into a presentation, and (3) playing out the presentation to the user. As a future step, we plan to extend this work into a distributed environment, where the multimedia data (i.e., segments) and related constraints reside on a server site and the tasks related to selection and organization of contents into a presentation are carried out by local hosts (clients). The server is responsible for providing the clients with the available metadata (i.e., inclusion/exclusion constraints, set of multimedia segments, and organization constraints) of a particular subject in multimedia upon a client's request as well as feeding the clients with the contents of the presentation (i.e., the set of selected segments). We think that these extensions are easier due to the use of open system concepts[2] in most of our design decisions.

Acknowledgments

We would like to thank...

Notes

1. How the segments are selected is explained in our previous work.
2. *An open system* is the one that can be incrementally extended with the addition of new functionality without disturbing the existing system components.

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, November 1983.
2. N. Audsley and A. Burns. Real-time system scheduling. Technical Report YOR 134, University of York, Department of Computer Science, 1990.
3. D. Bordwell and K. Thompson. *Film Art An Introduction*. McGraw-Hill Inc., 4 edition, 1993.
4. Thomas H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
5. Roger B. Dannenberg, Tom Neuendorffer, J. M. Newcomer, Dean Rubine, and David B. Anderson. Tactus: toolkit-level support for synchronized interactive multimedia. *Multimedia Systems*, 1:77–86, 1993.
6. G. Davenport, T.A. Smith, and N. Pincever. Cinematic primitives for multimedia. *IEEE Computer Graphics & Applications*, pages 67–74, July 1991.
7. D. Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.
8. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
9. Marc H. Graham. Issues in real-time data management. *Real-Time Systems*, 4(3):185–202, September 1992.

10. Veli Hakkoymaz and G. Ozsoyoglu. Automated assembly, organization and playout of multimedia presentations as a constraint-driven approach. Technical report, Case Western Reserve University, 1996. in preparation.
11. Petra Hoepner. Presentation scheduling of multimedia objects and its impact on network and operating system support. In *Network and Operating System Support for Digital Audio and Video*, 1991.
12. Petra Hoepner. Synchronizing the presentation of multimedia objects-oda extension. *ACM SIGOIS Bulletin*, 12(1):19–32, July 1991.
13. Huang-Cheng Kuo and Gultekin Ozsoyoglu. A framework for cooperative real-time transactions. In *Proceedings of The First International Workshop on Real-Time Databases: Issues and Applications*, March 1996.
14. T. D. C. Little. A framework for synchronous delivery of time-dependent multimedia data. *Multimedia Systems*, 1:87–94, 1993.
15. T.D.C. Little and A. Ghafoor. Interval-based conceptual models for time-dependent multimedia data. *IEEE Trans. on Knowledge and Data Engineering*, 5(4), August 1993.
16. C. L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, 1968.
17. C. Nicolaou. An architecture for real-time multimedia communication systems. *IEEE Journal on Selected Areas in Communications*, 8(3):391–400, April 1990.
18. G. Ozsoyoglu, V. Hakkoymaz, and J. Kraft. Automating the assembly of presentations from multimedia databases. *IEEE Int. Conf. on Data Engineering*, February 1996.
19. N.U. Qazi, Miae Woo, and Arif Ghafoor. A synchronization and communication model for distributed multimedia objects. *ACM Multimedia*, June 1993.
20. Krithi Ramamritham. Real-time databases. *International Journal of Distributed and Parallel Databases*, 1992.
21. A. Silberschatz, J. Peterson, and P. Galvin. *Operating System Concepts*. Addison-Wesley Publishing Company, third edition, 1991.
22. Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.
23. Jeffrey D. Ullman. *Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.
24. Taieb Znati and Brian Field. A network level channel abstraction for multimedia communication in real-time networks. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):590–599, August 1993.

Received Date
Accepted Date
Final Manuscript Date