CHAPTER IV
CHAPTER IV

Presentation of Multimedia Data

A multimedia presentation describes how a multimedia scenario (spatio-temporal structure) is exhibited to the user. The main objective of a presentation is to read specifications and generate output. The second is to maintain the QoS(Quality of Service) parameters within tolerance range and resynchronize if need be using methods like wait/slip and skip-ahead (drop) etc.

Fig. 4.1 shows time line for presentation of five media objects. The diagram indicates that the key concepts in a presentation are that of time during which a data unit is observable by the user and how the display of one data unit is synchronized with another piece of data.
In this chapter we propose a scheme which generates relative times of presentation of objects without passing the authoring effort to the user. This specification of relative timings is called presentation schedule. This schedule is stored as a table which is sorted in ascending order of start times before going to the playout phase. During the playout phase the objects are displayed.

This chapter is divided into two sections. The first section discusses some of the issues crucial for management of presentation. The second section gives a scheme to generate presentation schedules and present multimedia data.

4.1 Issues involved in presentation

In this section our discussion revolves around three key issues of presentation. One, the need for presentation management, and second, framework for providing presentation management, and lastly synchronization requirements.

4.1.1 Need for presentation management

A variety of situations like libraries, publication, departmental training and education cells use presentation management in multimedia application. Presentation is the user’s interface with the MMDBMS where the user is on the receiving end. Though appropriate presentation of data/results is also necessary for traditional data, but it becomes more crucial in case of multimedia data because of the following factors:
• **Multiplicity of data objects**: Multimedia data can be viewed as an integration of data from diverse media such as text, graphics, video etc. This multiplicity implies that, if the various media objects are to convey the desired information to the user, there must be a way to coordinating how and when they are to be presented, both in terms of space and time[25].

• **Spatial aspects**: Many multimedia objects may have to be displayed simultaneously on the output media, paper or screen. As a result while presenting, care needs to be taken about spatial co-ordinates of data. Objects should not overlap, if they are not intended to according to their specifications, and relative spacing needs to be maintained[49,125].

• **Temporal aspects**: Major problem of synchronizing multimedia data, specially continuous data can be traced to the timing constraints imposed on their presentation. The temporal aspect becomes crucial and may be a bottleneck in case of networks. Parameter such as jitter and skew(explained later in this section) have to kept under tolerance limits[10,153].

• **Asynchronous user interaction**: When the user is allowed to interact freely with the presentation, some new synchronization problems can arise, for example the user may like to change order or speed of presentation[165].
4.1.2 Framework for providing presentation management

Thimm and Klas[158] in their study have discussed extensively the framework for providing presentation management. Here we discuss some alternatives which have been suggested for the placement of playout management for multimedia databases[158].

Application based playout management:

In this case the playout is managed by an application program. Playback of the continuous data streams, coarse grained as well as lip synchronization, and the handling of the user interactions is hard-wired in the individual application. From the user's point of view, it makes no difference if, as an extension, the application program provides functionality, but this may be prone to errors besides causing workload on the application.

General presentation tool based playout management

In contrast to the above approach, multimedia playout management is performed according to a more general scheme. A general presentation tool that provides playout management functionality is provided along with the database management system. Differences between the presentation tool and the database management system are taken care of by an interface mapper. The latter is a database application program which maps requests of the presentation tool into
such a form, that they can be handled by the database management system. It also transforms answers back into a form manageable by the presentation tool. In this case, the playback of continuous data stream, their temporal synchronization, especially the lip synchronization, between the video and its audio-soundtrack, as well as the user interactions are handled by the generic functionality of the coupled presentation tool. This approach realizes a loose coupling of the presentation environment with the database management system and its services.

fig. 4.2a
Object-oriented multimedia database with playout management functionality

Object-oriented databases allow for encoding real world semantics of the application domain directly within the database. The methods of such an object provide the typical functionality to present the video clip, possibly reflecting synchronization constraints such as the lip synchrony, and to control the presentation, e.g. halt, restart, backward, fast-forward, go-to etc. with respect to our example, we see that both video clips, the audio-soundtrack as well as the image are available as such database objects. The presentation specification which
consists of messages issued to these objects and respective control structures is also modeled as a database object. At the time of presentation, this object invokes the specified methods at the appropriate points in time. Using this approach it is hard to meet the temporal requirements of multimedia presentations at runtime as well as the required responsiveness to user interactions. Playout management execution is achieved by the invoking methods which is handled by the central database management system environment.

Playout management as database management integrated service

In this arrangement, playout management functionality is provided as integrated service of the database management system. The underlying DBMS interprets the presentation specification by mapping it into corresponding playout management actions (e.g. synchronization enforcement actions). These actions are carried out by the database management system. This alternative provides playout management independency advantageous for many applications similar to the provision of independency from the physical data storage.

4.1.3 Synchronization requirements

One of the objectives of presentation is to maintain the Qos (quality of Service) parameters within tolerance limits and resynchronize if necessary. In this section we describe the Qos parameters required for synchronization. Quality of service (Qos) requirements for intrastream synchronization, requirements are
expressed in terms of two parameters: delay and jitter[140]. Delay refers to the maximum delay of a data unit of a stream between its input and output at the corresponding devices whereas jitter implies the variance of delay allowed (maximum delay - minimum delay)$^2$. The selection on the bounds of these parameters depend on the application used and the user involved. For example, low delay bounds will be suitable for a user requesting fast interactivity. For discrete streams like image and text, explicit jitter bounds are not specified, so the jitter is also bounded by the delay bound. But in some situations like computer animation sequences, it may be necessary to specify tighter jitter bounds as the rate of movement of frames of the animation sequence should be suitable for human viewing, neither too fast, nor too slow. Values of skew and jitter have been experimentally obtained[25,6].

Because of inter-media synchronization, the streams of dissimilar media(e.g. aural and visual) are to be synchronized and presented. Therefore, not only the individual media data has to be presented with acceptable quality but also, the media streams should appear to be coordinated with other media streams. For example some parts of a sound track may belong to specific animation sequence, subtitles should appear with specified frames in a video sequence, etc. In this case, the additional Qos parameter is skew. Skew is the difference between the stream time systems of the various data streams that make up the multimedia object. Jitter is noticed in a single stream whereas skew is noticed on an aggregate of multimedia streams and thus involves inter-stream synchronization. Skew can
result from a number of factors such as variation in delays experienced by different streams in the network, delays at the end systems, loss of data, clock variation between the source and destination machines, low quality of service specification etc. Like jitter, skew is also quantified in terms of skew bounds acceptable for an application based on the limits of human perception[147].

For specifying interstream synchronization requirements, the skew parameter is defined as:[140]

\[ \text{Skew}(P_1, P_2) \rightarrow (P_3, P_4) < \text{skew-bound.} \]

\( P_1, P_2, P_3 \) and \( P_4 \) denote observation points of data streams(i.e. input/output devices). Let \( D_1 \) denote an arbitrary data unit observed at time \( t_1 \) and \( D_3 \) a data unit observed at \( P_3 \) at time \( t_3 \). The same two data units are observed at \( P_2 \) and \( P_4 \) respectively at times \( t_2 \) and \( t_4 \). The skew expression from above states, that if between \( D_1 \) and \( D_3 \) a certain distance in time (\( d_1 \)) was observed at points \( P_1 \) and \( P_3 \), the distance (\( d_2 \)) between \( D_1 \) and \( D_3 \) as observed at points \( P_2 \) and \( P_4 \) does not differ from \( d_1 \) by more than skew-bound.

\[ \begin{align*}
  &t_1(D_1 \text{ at } P_1) & t_3(D_3 \text{ at } P_3) \\
  &\hline & \hline
  &\quad d_1 \quad & \quad \\
  &\hline & \hline
  &t_2(D_1 \text{ at } P_2) & t_4(D_3 \text{ at } P_4) \\
  &\hline & \hline
  &\quad d_2 \quad & \quad \text{skew} \\
\end{align*} \]

*fig. 4.3*
Skew can be measured with respect to real-time as an offset to some mutual presentation start time between the source and destination, or can be measured with respect to another stream for example, between audio and video as shown in the diagram above. Because many streams are possible, we may characterize both intermedia and real-time reference skew for k streams using matrix representation as skew = sk_{p,q} where sk_{p,q} describes the skew from stream p to stream q (q to p is negative) and the k_{th} element corresponds to a real-time reference. Interstream tolerance \( \Phi = \theta_{p,q} \) and the target skew matrix TSK = tsk_{p,q} which indicate tolerances and target values between streams can also be defined. The skew and tolerance matrices are stored in database as metadata. Skew best measures intermedia synchronization for continuous media. For characterization of discrete events associated with timed playout of text, graphics and still images, we can apply real-time scheduling terminology as already mentioned (e.g. maximum and minimum delay).

In the fig. 4.4 below a motion picture composed of blocks of audio and video is shown. If the skew between the presentation unit of audio and video is within the tolerance limits, the picture is appropriate for human viewing else resynchronization by dropping or duplicating frames of audio or video is needed.
For intra-object synchronization, presentation requirements comprise of the accuracy concerning delays in the presentation of LDUs (Logical Data Unit, which is a presentation unit) and for inter-object synchronization, the accuracy in the parallel presentation of media objects. The tolerance of data to skew and jitter during playout varies widely depending on the media. Audio and video require tight bounds of the order of hundred of milliseconds, whereas synchronous text and images can tolerate skew on the order of seconds. Furthermore, audio and video can tolerate different absolute timing requirements during playout as the human ear can discern dropouts in audio data more readily.

### 4.2 Presentation management

In the present section we describe a scheme for generating a synchronized presentation plan/schedule which is generated at presentation realization time from the document level specification.
This scheme takes as input the temporal specifications already captured and stored, and generates a deadline table as the output. A generated schedule/plan is represented as a deadline table which provides a logical time axis and it contains the identifiers of the data to be presented. This deadline table gives the relative timings of start and finish of display of a component of a multimedia document.

We model the concept of presentation, which includes the time during which and image(or a video) is observable by the user, but also which expresses the fact that the image display should be synchronized with another piece of data, either in sequence or in parallel; for example, a slides show. Temporal relations before, after, parallel start, overlap, during etc. have also been defined in terms of triggers and specified within the database schema on the class level and used for each object presentation[144]. A document representing an object class is represented as a tree including temporal relations and is defined in the schema[143].

The presentation schedules for the objects are generated by traversing the tree and using the temporal relations defined previously. The user can then start running the playback engine which will read the specifications from the deadline table and retrieve the data and produce appropriate presentation output. The presentation management is divided into two phases:

(1) pre-play management

(2) playout and control
4.2.1 Pre-play management

By this stage the active objects have been identified (active objects are the ones which participate in presentation). We assume that pointer to the root node of the active document is available and thus all the subnodes are also accessible. (For our purpose we assume that we get an instance tree otherwise from a class tree it is possible to get an instance tree[129]).

We calculate the deadlines of the start and begin of each subobjects of the document. When the document is created the temporal relations are also stored along with it. There are various data structures associated with each data base and templates are one of them. Temporal relations have been defined in terms of triggers and are used after being instantiated as explained in chapter III. In the previous chapter we have given algorithm treetraversal which creates a link list called templatelist. The nodes of this link list are in form of source/destination pairs. Each node of the templatelist looks like:

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
</tr>
</thead>
<tbody>
<tr>
<td>trig[1,1]</td>
<td>trig[1,2]</td>
</tr>
<tr>
<td>trig[2,1]</td>
<td>trig[2,2]</td>
</tr>
<tr>
<td>nxt</td>
<td></td>
</tr>
</tbody>
</table>

*fig. 4.5*
id₁ is the object identifier of the first object (source); id₂ is object identifier for the second object (destination).

trig[1,1] gives the (id, event) pair which is begin trigger for object id₁.

trig[1,2] gives the (id, event) pair which is begin trigger for object id₂.

trig[2,1] gives the (id, event) pair which is end trigger for object id₁.

trig[2,2] gives the (id, event) pair which is end trigger for object id₂.

event can take values begin or end.

nxtis the pointer to the node of the same type.

4.2.1.1 Creation of nodelist

From the templist we create a nodewise list and we call it nodelist. This process is carried out by algo createnodelist which involves scanning of the templist created by algo treetraversal for each node id and setting the value of appropriate ptr[i,j] by src or dest id. Value of i is given by the value of row of array trig where the id appears and the value of column depends on the event associated with the id, i.e. for begin j is set to 1 and for end j is set to 2. Thus we get a list with a node corresponding to each id. The algorithm for creating nodelist is given below[142]:
algo createnodelist(start)

/*this algorithm takes as input a pointer start which is a pointer to the beginning of the templist and produces a nodelist given in fig.4.7*/

for id = 1 to n /*n is the total number of nodes in the document tree*/
{
    sptr = start
    /*search for id in all the nodes of templist*/
    while sptr <> null
        {for i = 1 to 2
            for j = 1 to 2
                /*search is carried out in array trig which is part of node*/
                if id = sptr -> trig[i][j] then { flag = 1
                                                    break
                                                }
        } /*if search is successful end for*/

    if flag = 1 then
    {
        if j = 1 then freshid = sptr -> src
            else /*if j=2*/ freshid = sptr -> dest
            /*store event associated with the id searched*/
            eventtype = sptr -> event

            /*createnode creates a node to be appended to appropriateptr[i,j] list and returns a pointer to the node*/

            Cnfresh = createnode(freshid)
            if eventtype = beg then set col = 1
            else/*eventtype = end*/ set col = 2
        }
    }
}
row = i /* row of ptr array is set equal to row of array
trig where id search was successful*/

appendnode(CNfresh, row, col)

/* appendnode attaches the node with created in
proper ptr list, by using values of row and col*/

sptr = sptr -> nxt /* move to next node of templist for searching
id*/

}/* end while*/

} /* end for */

end algo

The structure of a node of this nodelist created by algorithm createnodelist is
explained in fig.4.6.

\[
\begin{array}{|c|c|}
\hline
\text{id} & \text{beg} & \text{end} \\
\hline
\text{ptr[1,1]} & \text{ptr[1,2]} & \text{ptrnx} \\
\hline
\text{ptr[2,1]} & \text{ptr[2,2]} & \\
\hline
\end{array}
\]

\text{fig.4.6}

For each node we store a list of all other nodes which have any kind of
synchronization relation with this node with identifier id. Each node of this list
consists of an object id and a 2 x 2 array. The elements of these array are further
pointers to link list.

ptr[1,1] points to a list of all those nodes for which id.begin is a begin trigger

ptr[1,2] points to a list of all those nodes for which id.end is a begin trigger

ptr[2,1] points to a list of all those nodes for which id.begin is a end trigger

ptr[2,2] points to a list of all those nodes for which id.end is a end trigger

Thus, the first column refers to the begin event of the object for which this node is
created and the second column refers to the end event pertaining to this object. In
this nodelist id is the source of the synchronization. The destinations of
synchronization are stored in four link lists pointed to by ptr[1,1], ptr[1,2],
ptr[2,1], ptr[2,2]. The first row refers to such nodes whose beginnings are
triggered by the node id (whether its beginning or end); and the second row refers
to those nodes whose ends are triggered by node id (whether beginning or end).
ptrx refers to the next such node in the link list. This nodelist consists of as many
number of nodes as the number of objects comprising the document[142].
Applying algorithm createnodelist on templist shown in fig.3.18 of chapter III results in the following nodelist:

**fig.4.7**

In the link list given above in fig.4.7, the number on the top of a node gives the object identifier (id or objid). For instance, in the first node id = 1. Beginning of node 1 (i.e., node with id = 1) triggers beginning of node 2 (because 2 appears in
the first row and first column of the array of pointers). Similarly for the second
device, beginning of node triggers beginning of node 5 and end of node 2 triggers
beginning of node 3. For node 4, beginning of node 4 triggers beginning of node 9
and end of node 4 triggers end of node 1. This way the complete list can be
interpreted. Thus the source of synchronization is written on the top and events
synchronized with it (i.e. destinations) are given in form of 2 x 2 array, each
element of which is a list of pointers as described in fig. 4.6.

4.2.1.2 Creation of deadline table

Traversal of this nodelist in preorder of nodes as given in the document tree
enables us to create a deadline table. The starttime, deadline and another deadline
are set using the information given in each node of the nodelist. Thus for
multimedia documents, the playout deadline for each node (component media
item) is calculated and stored in a table. The deadline-based representation of
temporal relation is unique for each document. The format of deadline table is
shown below[142]:

<table>
<thead>
<tr>
<th>ObjId</th>
<th>Starttime</th>
<th>Deadline</th>
<th>anotherdeadline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*fig.4.8*
**Objid** gives the unique object identifier; **starttime** gives the time when the object with objid should start displaying; **deadline** gives the time when the display of the object should stop; **anotherdeadline** is same as deadline fields in case of terminal nodes(leafs) but in case of non-terminal nodes(composites) this field may be different than the deadline because the duration mentioned in the composite may not exactly be same as that of the duration elapsed when its components are displayed related by the field tr. In the playout phase we take corrective actions to take care of this discrepancy[142].

For creating the deadline table we again use the begin and the end triggers from the nodelist to update start/endtime. If an objid.begin is begin trigger for some nodes then the start times of all such nodes are set equal to the start time of this node(after taking care of the relative offsets). If objid.begin is end trigger of some nodes then the endtimes of all such nodes are set equal to starttime of this node. similar updations are done for objid.end. The objid in the above table is pointer to the particular object and for continuous media it is pointing to the first frame in the sequence.

Now we sort this table on starttime and get a table which has start times arranged in increasing order because we require an ordered sequence component of objects to be sent to playout phase[142,144].
A detailed algorithm for the steps explained for the calculation of deadlines is shown below:

**algorithm updatedeadline**(start, root);

/*this algorithm takes as input the nodelist and the document structure for details of duration and offset. The output is the deadline table. start points to the beginning of the nodelist and root is the root of the document tree*/

thisnode: = root;

/*starttime of root is set = 0*/

starttime(thisnode) : = 0;

while thisnode <> null

    /*a node which is a parent, i.e. non-leaf is pushed in a stack*/

    if thisnode is non-terminal then push(S,thisnode)

    /*times of all the events triggered by this node’s beginning(i.e. thosenode.beg/end) are set*/

    for all thosenodes for which thisnode(objid.begin) is begin trigger do
        update-starttime(thosenodes)

    for all thosenodes for which thisnode(objid.begin) is end trigger
        update-endtime(thosenodes)

    /*for terminal node deadline and anotherdeadline are set*/

    if thisnode is leaf

        deadline(thisnode) : = starttime(thisnode) + duration(thisnode);

        anotherdeadline(thisnode) : = deadline(thisnode);

    /*times of all events triggered by end of this node are set*/
for all those nodes for which thisnode(objid.end) is begin trigger do
    update-starttime(thosenodes)
for all those nodes for which thisnode(objid.end) is end trigger do
    update-endtime(thosenodes)

/*another deadline is set for the parents*/
if thisnode is the last leaf(last sibling)
    thisnode = pop(S);
    anotherdeadline(thisnode) = starttime(thisnode) + duration(thisnode);
/*set times of events triggered by end of the nodes popped*/
for all those node for which thisnode(objid.end) is begin trigger do
    update-starttime(thosenodes)
for all those nodes for which thisnode(objid.end) is end trigger do
    update-endtime(thosenodes);

/*obtain the next node in pre-order*/
thisnode = traverse-tree;

end algo

Procedure traverse-tree traverses the document tree in preorder; push stores the non-leafs in a stack; pop retrieves the last parent stored in the stack; update-starttime and update-endtime set the values of start time and deadline( after taking care of the difference between deadline and another deadline values), respectively in the deadline table; those nodes are obtained by scanning the link lists pointed to by the array elements of ptr of the node corresponding to id of thisnode.
The deadline table obtained after updatedeadline algorithm with input as start and root (where start is the pointer to the beginning of nodelist in fig. 4.7 and root is the pointer to root of document tree of fig. 3.17) is given below:

```
<table>
<thead>
<tr>
<th>Objid</th>
<th>Starttime</th>
<th>Deadline</th>
<th>Anotherdeadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>14</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>
```

We now explain how the entries of the deadline are obtained. Initially this node = 1 and is pushed into the stack and its starttime = 0 is obtained. From the nodelist we see that 1 beg is begin trigger for 2. Therefore, starttime of node 2 = 0. Because 2 is also a non-terminal node, it is pushed into the stack. Next the value of this node
is 2 as obtained by traverse-tree. From the nodelist we see that that 2.beg is begin trigger for 5, so starttime of 5 is set equal to starttime of 2 i.e., 0. Again traverse-tree is call and returns thisnode = 5. The nodelist indicates that 5.end is begin trigger for 6. Now 5 is a terminal node so its deadline is calculated as 0(starttime of 5 as set in the previous step) + 3(duration of 5 as obtained from the document tree) = 3. Also anotherdeadline of 5 is set to 3, because 5 is a leaf node. Because 5.end is begin trigger for 6, therefore starttime of 6 is set = 3+1 = 4 (end time of 5 +offset as given in document tree). Next, traverse-tree is called and returns thisnode as 6. 6.beg does not trigger anything so no updations are made. The algorithm then calculates end time of 6 = 4 +4 = 8(starttime of 6 + duration of 6), and subsequently deadline of 2 is set as 8 because 6.end is end trigger for 2. Also 6 is the last child of its parent. So the stack is popped and we set anotherdeadline for node 2 = starttime of 2 +duration of 2(as given in document structure) = 0 +8.

In case of node 2 deadline and anotherdeadline are same. The process continues and the next subtree to be processed is with parent 3. The starttime of 4 which is to follow node 3 is set to anotherdeadline of node 3 and not 12, which is the deadline of node 3. This difference arises because the duration of composite node 3 is not same as those of its components related by tr given in 3.

Though the deadlines are calculated for all the nodes in the tree it is obvious that only the leaf nodes are displayed by the playout management. It was necessary to update the deadlines for all nodes at all levels in order to arrive at accurate
deadlines (both start and end) of the leaf nodes. We assume here the retrieval times to be negligible as compared to playout times. Also for the time being we are ignoring network delays (we are considering a local model).

This finishes the pre-playout processing. The preplayout process schedules the presentation of periodic objects such as texts and graphics; and gives the starttime for the playout of segments of periodic streams of audio and video. Once the starttime (start deadline) for a continuous is obtained there is no need to calculate start and deadlines for the subsequent frames as the frames are assumed to arrive in sequence and each frame takes one logical time unit (LTU) for display. E.g a video can have an initial and final playout i.e. start and finish but its component frames can be played with respect to their initial deadlines and their sequence [6]. The (finish) deadline is required for continuous media because it may be followed by other objects whose start time depends on the former’s finish.

4.2.1.3 Using presentation schedule for retrieval

Besides the playout phase, presentation schedules may be used to extend database system’s interface by providing the ability to order the objects for retrieval and delivery. It can enable database system to schedule both current and near-future object retrievals. Presentation schedules are helpful to plan object retrieval in two ways. One, to reduce latency and second, to avoid resource wastage.
Latency reduction minimizes the time between the user's request and the corresponding delivery. It can be achieved by providing the database system with a prior knowledge of future accesses by means of a schedule. The database system is then able to pre-deliver multimedia content objects. This is explained with the help of fig. 4.10. The requirements are shown in fig. 4.10a, fig. 4.10b shows the time gap occurring due to latency time after the conclusion of the video. The latency is reduced/removed in fig. 4.10c since the database system knows which item is needed next. So, the database system pre-delivers video2. It is to be noticed that the delivery time remains the same.

![Diagram showing time, Video1, and Video2](image-url)
Presentation schedules can also be used to achieve reduction in the waste of system resources. This can be done by stopping the delivery of multimedia content to prevent excess delivery of some media item. Let us consider a situation where two pieces of multimedia objects of different playout durations are needed. The objects shown in fig.4.11(a) are an audio clip of length 25s and a video clip of
length 20 seconds. The user wants to stop the audio clip when the video clip ends. With the traditional request model, the stop command to the database is sent when the video ends. But, due to the latency of the request, the database system will keep delivering data for latency period. This results in unnecessary wastage of resources as shown in 4.11(b). With a presentation schedule the database system stops the audio delivery before the video ends, as shown in 4.11(c), and avoids the waste of system resources by sending request to end audio a little earlier than finish time of video.

![Diagram](fig.4.11(a))

![Diagram](fig.4.11(b))
The database system has limited resources available to meet user's requests. Therefore, presentation schedules by giving prior information enables the database system to pre-deliver content to balance the load. Presentation schedules provide database system with a sequence of requests. This allows the database system to create a time-based access plan for the retrieval of the requested content objects. For this presentation schedules must be written using an appropriate syntax and parsed. Similarly, presentation schedules provide a prior knowledge about requests allowing the database system to optimize access and delivery.

4.2.2 The playout management

The pre-play management creates a presentation schedule for multimedia objects in form of deadline table, now the problem of synchronization reduces to ensuring that objects are presented at their scheduled times as given by the deadline table.
The purpose of the playout process is to send the objects (individual components of the document) to their destination (output device) based on their starttimes. The output devices consume or present this data till the deadline (endtime) is reached. This playout process schedules the presentation of static events such as text and graphic displays and initiate the playout of segments of dynamic streams of audio and video. For continuous data the stream management is more critical since playout deadlines for the presentation units (e.g., a frame with respect to a video) have to be considered. Once the order is established, the playing of these segments relies on their relative sequence numbers rather than individual deadlines associated with each frame, as explained above. For example, a motion picture scene can have an initial playout initiation deadline, but its component frames can be played out with respect to its initial deadline and their individual sequence numbers. Deadline table is arranged in increasing order of starttime and thus objects are assumed to arrive in sequence.

Playout is accomplished by mappings (for example a raw bitmap file might be copied to the pixels in the frame buffer) which are chosen from display methods (there is display method for each object class which is defined along with the object class) available in the appropriate data definition class where we have defined a set of parameters for each object class [143]. These parameters vary with the media type and may be defined statically by the attribute list or may be obtained dynamically by a function evaluation. We use the former technique.
Other elements like playback rate and direction (in case of continuous media) can also be considered as parameters of the play. As a result, instead of defining new presentation, we can use the same temporal structure and only the parameter speed varies when users require presentation of data with varying speeds.

Multimedia presentations are inherently dependent on hardware, for example, it is not possible to have audio playback without speakers and colored compositions can be displayed only on colored monitor. At the presentation realization time the system-specific concerns cannot be totally ignored, but at the same time the structural and temporal aspects of the document to be presented should be portable. In order to properly describe presentation of an object with an associated temporal structure, presentation module must have access to system related information like the list of output devices available, function of the user-interface library and the range (maximum and minimum values) of their attributes. Presentation should be specified in a way which is independent of hardware, so that it can be run on various platforms. If we separate presentation specifications from system specifications, it is possible to make presentations reusable. Waynblatt[175] suggested the concept of placeholders to accomplish this. These are dummy values which are used in place of hardware specific values and functions. We use the idea of placeholders "#" to refer to hardware devices when a presentation is loaded. The playback device uses the local database of system specifications to determine the correct value or function and substitutes the
appropriate values instead of placeholders. Presentation also assigns playback rate, duration to each presentation unit. The format for playon() looks like:

\[
\text{Play(presentation name, } \#, \text{ device no., mode, start, direction, duration)}
\]

To avoid defining new presentations which are similar we provide for modes. For instance, for video presentation mode can be normal, slow or fast. A combination of modes and directions will give fast-forward, fast-reverse etc.

Now we apply the playback loop to the deadline table. The playback loop checks and enforces synchronization according to temporal constraints. Playout management also resolves the conflict in case duration of the composite does not match with the duration of its components. Instead of calling it just playout we prefer to call it playout loop because it goes on scanning the deadline table in a circular fashion till all the objects have been displayed. In each cycle for each objid the starttime is checked versus the clock. The inter-object synchronization constraints have already been taken care of while arriving at the sorted deadline table. For a static object like text and images the play-on() function is called when the clock reaches the start time for the particular objid and once the play-on() is activated, it only has to check that the image has not exceeded its allotted duration. The allotted duration is calculated on the basis of deadline and another-deadline, whichever is maximum. If deadline was less than another-deadline the silence() method is invoked. This constitutes the control of presentation. For
continuous or dynamic media, like video an audio, objects are fed to the playback buffer ahead of time and then played. The algorithm keeps adding frames to the buffer till the queue is empty. We assume that the audio/video objects are arranged in a queue of frames. Each frame during playout takes 1 LTU. The playout continues till the buffer empties. The duration specified by the deadlines is checked versus the clock. There are three possibilities:

1. the buffer empties first - invoke silence() till the max \{deadline, another-deadline\} is reached

2. buffer is not empty and max\{deadline,anotherdeadline\} is reached - discard the remaining frames in the buffer

3. third case is ideal - nothing needs to be done

The playout algorithm for managing presentation deadline is outlined below.

algorithm playout(deadline table)

/*this algorithm takes as input the deadline table and sends the objects to the playout device according to their starttimes*/

/*all the objects in the table are scanned one by one*/

while i <= n do

/*for static objects clocktime is checked vs the starttime of the object in the deadline table. If the starttime has been reached and the object is not already being displayed then start display*/
if objid[i] is static then

if clock \( \geq \) objid[starttime] and flag = 0 then

\[ \text{flag} := 1 \]

/*duration of the play is calculated*/

\[ \text{min} = \min\{ \text{objid(deadline)}, \text{objid(another deadline)} \} \]

\[ \text{dur} := \text{min} - \text{starttime} \]

/*send objid to destination with appended deadline*/

play-on(objid, #devname, mode, 0,direction, dur)

/*0 is the value of start time, direction is immaterial in case of static objects*/

\[ \text{diff} := \text{another deadline} - \text{deadline} \]

/*conflict of duration between parent and children is resolved using the silence() function*/

if diff > 0 then invoke silence( ) for type(objid.lastchild) for remaining time i.e. diff

/*processing for the dynamic objects proceeds in the similar manner*/

else if objid[i] is dynamic

\[ \text{min} = \min\{ \text{objid(deadline)}, \text{objid(another deadline)} \} \]

/*time period during which object is played is calculated*/

while objid[starttime] \( \leq \) clock \( \leq \) min or objid[starttime] + LTU

* no. of frames

/* send next frame to playout buffer */
play-on(nextframe, name, mode, #devname)

/*synchronization is achieved by playing silence() */

if queue(empty) and clock <= max then invoke silence( )
else if queue(not empty) then drop remaining frames /* frames left*/

end while

end algo

The parameters for continuous data types should be checked before the playback of each segment to determine the value of playback parameters whereas for static datatypes (i.e. single segment) they are checked periodically to allow variation in the presentation parameters as well.

For periodic data such as audio and video, data can be lost resulting in dropouts or gaps in playout. Such losses cause the stream of frames to advance in time or cause a stream lead. Similarly, if a data frame is duplicated (or silence invoked), it causes the stream to retard in time or a stream lag. By dropping or duplication frames it is possible to control the rate of playback. Initiation of frame dropouts is permissible for audio and video which have substantial short-term temporal data redundancy. However, for text and graphics, this is not true as their tolerance to delay is greater.
We have define another object type clock, modeled like a stopwatch. This object consists of two attributes. First, time, to measure the elapsed ticks and second, time unit, which defines the unit represented by each clock tick. For any given media stream, this time unit (LTU - Logical Time Unit) is the amount of time required to present an atomic data unit i.e. the smallest data item such as a video frame of an audio sample. Each atomic data unit has a presentation duration equivalent to an atomic interval (e.g. 1/30s for a video frame). The atomic interval is the same for any given media type and is the smallest interval recognized in the schedule. Thus the presentation duration of any stream can be expressed as a multiple of the atomic interval of the media type making up the stream. With our representation, multiple streams can be accommodated using multiple temporal relations. The time instants making up the intervals represent relative points in time during which the particular media stream will be presented. With only the knowledge of the presentation time of the first object, the presentation times of all other objects can be calculated based on their temporal relations. This further implies that the particular time instant at which a presentation can start can easily be modified without affecting the entire schedule. This feature is central to the modeling of user interaction since we can modify the presentation interval of a given multimedia object without needing to modify the overall presentation schedule. The method associated with object type is getelapstim which measures the time elapsed since a specific instant which helps to calculate durations of the play.
As playout time durations increase, the system delays become less significant warranting the development of ideal presentation algorithms. Images, text and graphic segments can be effectively synchronized with such an ideal scheme because their presentation durations are large (greater than a few seconds), whereas video and audio frames have a very short presentation interval (less than 1s) and require specific resource scheduling.

In this chapter we have proposed a scheme which generates presentation schedules without passing the authoring effort to the user. Presentation schedule is in the form of a deadline table. The values of the parameters of this table are calculated using the specifications already given in the document structure. Taking a presentation deadline table as input, multimedia playout management controls presentation realization phase i.e. the period of time from the request till presentation is complete. Presentation schedules can also extend database system’s interface by providing the ability to define, select, and manage objects for retrieval and delivery. It enables database system to schedule both current and near-future object retrievals.