A Multimedia Temporal Specification Model and Language

Paul Pazandak
Jaideep Srivastava

Abstract

Several models support the specification of temporal relationships for the recording and presentation of multimedia. The five principal approaches used by these models include hierarchic, timeline, reference point, event, and event-based languages. Many of the current models, however, lack expressiveness, or require complex specifications. We introduce a temporal event-based model that defines three basic relations that express causality, deferment, and fine-grain synchrony; within this model we introduce the notions of deferment, and affectable and non-affectable events. The language based upon our model contains just three primitives for temporal specification of these relations. Using them we can express the temporal relations of the sixteen models studied, and we can further express several relations that cannot be specified by these models. An interesting feature of our model is that it supports both a conceptual specification as well as extensions for defining the system implementations, resulting in additional flexibility. We describe our model using a temporal specification model framework we used previously to compare several models.

1. Introduction

The demand for, and use of multimedia is growing rapidly. One area of particular importance is the development of applications that support the creation of multimedia presentations: the organized playback or recording of multimedia information. A multimedia presentation contains timing information (a temporal ordering) describing *when* (and perhaps how) each media object should be presented[1]. The set of temporal orderings is called a *temporal specification* ; its structure is regulated by a temporal specification model. A temporal ordering may be absolute (e.g., play video X at 2:00, play audio Y at 3:00), or relative (play audio Y *after* video X) to other media objects being presented. This temporal ordering must be specified using some mechanism, such as

[1]We use variations of 'presentation' in the abstract or typeless sense -- the presentation of media objects will actually depend upon the data type of the objects -- e.g., generally, audio is *played* on a speaker, while video is *displayed* on a screen.

a scripting language, provided by the application. There is a wide range of proposed (and implemented) temporal specification models to date. Two primary discriminators are: 1) the *expressibility* of the model, which is the types and number of temporal orderings that can be expressed in a model, and 2) the *simplicity* of the model, meaning how simple it is to define a temporal specification (e.g., does it require learning an entire language to specify, or only a few predicates). Most current models are either too complex, or not expressive enough (the current models do not support negative delays within their specifications). For example, without negative delays, the following temporal specification cannot be expressed: *"Start the audio introduction two minutes before the main video presentation begins"*, without explicitly stating when the main video presentation begins.

To reiterate, a temporal specification model facilitates the ability to specify temporal orderings by defining *temporal relationships* involving media objects. The usual examples of media objects include images, graphics, movies/videos, text and audio soundtracks. Spatial specification (beyond the scope of this paper) is used to define the spatial relationships and attributes for presentations of media objects, such as the dimensions and location of a window on a monitor that will contain the playback of a video.

Temporal specifications are used directly or indirectly by applications to determine the sequence in which to record, or to present, media objects. Two common examples of temporal relationships are "parallel" and "serial". Two media objects having a "serial" relationship are played one after the other, while a "parallel" relationship will cause them to be played at the same time. These relationships temporally synchronize the presentations of media objects, synchronizing the starting, and perhaps the ending of media object presentations to other media objects' starts or ends; to some point in time; or, perhaps to the occurrence of some event.

In addition, a finer level of synchronization[2] is required to coordinate the presentations of media objects that have a high degree of *temporal interdependency*. Finer degrees of synchronization specified between media objects produces the synchronized presentation of related portions of the media objects. At the level of our discussion, media objects can be subdivided many times into *subobjects*, such as frames of a video, lines of a frame, and pixels of a line. Although not mandatory, the units of synchronization for presentation purposes are usually limited to logical units (or *media atoms*) such as frames, and samples of audio. An example is the synchronized playout of a video and its related audio. When a given frame$_i$ of the video is presented, the interdependent audio samples of the audio are also presented. We will use the term *fine-grain* synchronization to describe this degree of synchronization, as it best corresponds to our view that synchronization involves the coordinated presentation of finer granules, or atoms, of media objects.

Synchronization at this level is required, for example, when the audio contains the sounds visually produced within the video -- such as people talking -- so that we see the actions at approximately the same time as we hear the sounds. To avoid the "lip-synch problem", the delay between the playout of each video frame and its associated audio samples should be no greater than 150ms[2]. Multi-channel audio will require smaller delays, while a slide presentation with background music may not require fine-grain synchronization at all.

A useful way to describe the features of different models is to categorize them with respect to how the media objects are temporally synchronized. In [6], Blakowski defined three primary approaches for temporal synchronization:

[2]Invariably different terminology is used by different models. For example, *continuous* synchronization is used [1, 2], *total* synchronization in [3], *temporal composition* in [4], *tightly coupled inter-object* synchronization in [5], and *fine-grain* synchronization elsewhere.

- **hierarchical** - Using a tree structure, temporal relations are constructed using internals nodes (generally either 'parallel' or 'serial' temporal relation operators), and leaf nodes (media objects).

- **timeline** - Temporal relations are specified by indicating the start (and perhaps end) times of the media object presentations using implicit or explicit timelines.

- **reference points** (synch points) - The temporal relations are defined by logically connecting together the synchronizing points that have been inserted within the different media objects.

Two other approaches have also been used: **Causality modeling** (event-based[3]) and **programming languages** (event-based languages). Both of these involve synchronizing the begin/end points (and perhaps any point in between) of a media object's presentation to the occurrence of events. Events include system events, application events, user-defined events and events associated with the beginning or ending of a media object's presentation. Many of the models based upon each of these five approaches cannot directly express complex temporal relationships (e.g. timeline), or they require generating complex specifications (e.g. event-based languages).

We have defined a framework [7] that summarizes and compares several temporal models. It is apparent from our comparisons that models based on timeline and hierarchic approaches can express fewer temporal relations than models using the event-based approach. Using this framework we will describe our model for temporal specification. The model is based upon three simple, yet powerful relations for expressing *causality*, *deferment* and *fine-grain synchrony*. We

[3]The synchronization points used by Blakowski [6] for example, have some similarity to events, but are not the same.

have shown that the primitives of our language based upon these relations can express all temporal relationships expressible in any of the other models (see [7]).

In section 2 we review this framework, and follow with the description of our model in section 3. Finally, we discuss implementation issues in the form of system-defined extensions in section 4, followed by conclusions in section 5.

2. A Multimedia Temporal Specification Model Framework

Our framework provides a means of describing multimedia temporal models in a uniform manner. At the most basic level, every temporal model includes objects and relations. Temporal specification provides some means to express temporal relations between objects; a relation represents a temporal synchronization *intention*. A temporal specification defines the intended presentation of the media objects, not the actual presentation that will depend upon resource availability, data loss, and user interaction for example.

The richness of a model is directly associated with the set of objects and types of relations that can be expressed. In addition, several of the proposed models provide other capabilities and system supported features for enhancement through the use of programming languages. Rather than focus on the language in which it may be embedded, our framework focuses on the temporal objects and relationships, providing an accurate means of evaluating the expressiveness of a model.

2.1 Temporal Objects

In our framework, temporal objects are the model elements upon which relations are specified. Temporal objects include media objects, events and timepoints.

2.1.1 Media Objects

Within an object-oriented environment, media objects may be modeled in different ways and have different meanings. Several multimedia data models have been proposed in [8-12]. We define media objects as instances of media classes that represent both time-dependent (continuous), and time-independent (non-continuous) data types that may be used within multimedia systems. Examples of time-dependent data types are video, and audio; examples of time-independent data types are images and text.

Media objects can be simple, such as a video clip, or they can be composite media objects, also called multimedia objects, which are composed of simple and composite media objects. They can also be the atoms of simple media objects, such as samples of an audio clip. (Composite media objects may be composed of media objects of several different data types.) In general, relations are defined on media objects with respect to their entire interval, called *interval specification*, or, with respect to their endpoints, called *endpoint specification*. Endpoints are defined as the points that bound an interval, namely its starting point and ending point. Intervals have endpoints and an associated duration, and are used to represent the presentation (runtime execution) of media objects.

Current timeline models (e.g., [13-15]) and hierarchic models (e.g., [16, 17]) use interval specification. Allen's work [18] also discusses relations on intervals within his conceptual level specification. Endpoint synchronization has been used in synch point models (e.g., [2, 3, 6]), event-based models (e.g., [19, 20]), and event-based languages (e.g., [1, 8, 21]). At a conceptual level, endpoint relations have been defined in [22].

In addition, we can classify media objects as either having a *predictable* or *unpredictable duration* (e.g., [6, 19]). We define duration as the length of time it takes to complete the presentation of a media object once it has begun. This duration can be approximated by multiplying the number of

media atoms (e.g. video frames) within a time-dependent media object by its associated rate of playback (consumption rate). All temporal models can accommodate media objects of known or predictable duration. Examples include most pre-recorded media, such as films, and soundtracks. Unpredictable durations may involve media objects having one of the following: no inherent consumption rate, such as images; a flexible consumption rate, such as text (consumption rate is equal to the scroll rate and may be user-dependent); or, media objects that are being recorded in real-time, such as the signal from a surveillance camera.

2.1.2 Events

As stated above, the endpoints of media objects can be tied to events. Specifically, the beginning of an interval has a *begin* event, while the end of an interval has an *end* event. To provide additional flexibility, events can be defined at a *finer grain* within media and multimedia objects. For example, we could attach an event to the beginning of a particular sample associated with the start of a word within an audio clip. This is easily done when a flexible data model has been defined.

User- and *application*-defined events are additional temporal objects that can be used within a temporal specification. They can be part of temporal specifications involving other events, such as begin and end events. For example, we could indicate that a begin event for a media object should coincide with some application-generated (or system-generated) event, such as a user selecting a specific menu option from a graphical menu. We may also be able to define events, called *time events,* to be generated at specific times with respect to internally- or externally-defined clocks.

Several models (including all event models) provide the capability to specify that an event should be triggered or caused to occur, such as the starting or ending of a presentation. If an event can be forced to occur at any desired time, we say that it is an *affectable event;* otherwise, it is a *non-*

affectable event. Begin and end events are affectable, because we can cause them to happen (we can specify when the presentation of a movie should begin, and when it should be terminated) . Some user- and application-defined events may be affectable (such as opening and closing of graphic windows on a display), while others may be unaffectable (such as receiving messages or interrupts). Time events[4] are unaffectable, because we cannot cause an instant associated with a time event to occur; it will only occur at its defined instant in time.

Additionally, if our system provides image or sound analysis (and pattern-matching), we may be able to define events associated with the occurrence of a particular sound or image within the audio or video presentation. One could associate an event with the appearance of a red balloon, for example. This is an unaffectable event, since intuitively we cannot *force* the red balloon to appear in the video. Fujikawa [20] discusses the automatic generation of events corresponding to spontaneous actions within media objects, such as video.

To determine whether an event is affectable or not, we define a system boundary within which we have control over events, and outside of which we do not (see figure 0). For example, the causes of the following events perceived by our system exist outside of the system: 1) the progression of time, 2) interrupt generation, and 3) messages that are received (others also exist). We can use the occurrence of *any* event to trigger or cause the occurrence of an affectable event that resides within the system.

[4]It may be useful to define local clock time events as affectable events. We could then synchronize local clocks with other local clocks, system clocks, or world clocks, for example.

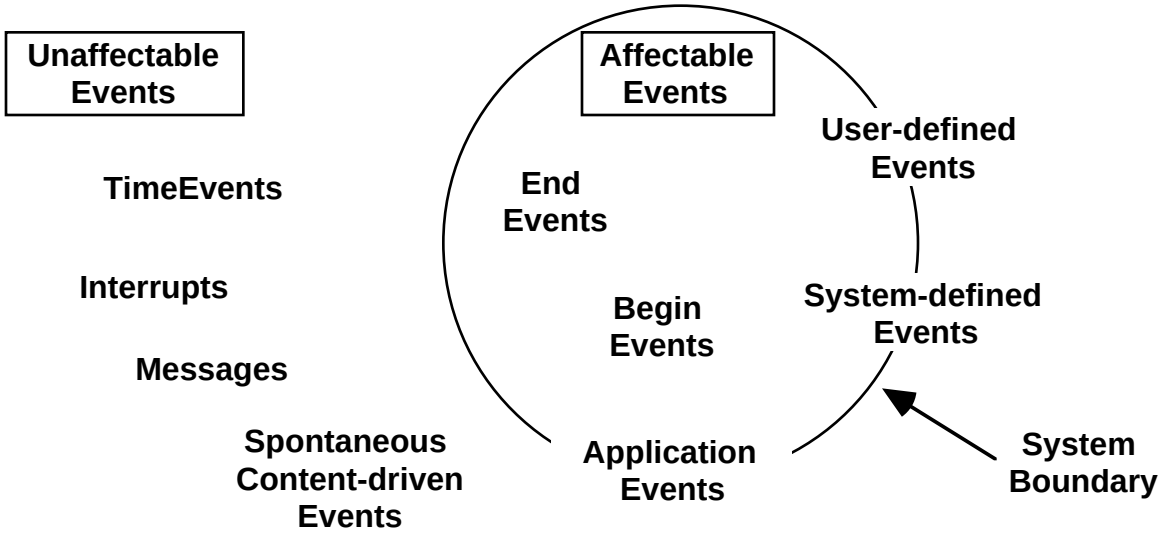

Figure 0. System boundary for affectable and non-affectable events.

2.1.3 Time

Timepoints are instants in time associated with user-defined timelines, i.e., they have no duration. Timepoints are used to specify starting and/or stopping points for media objects. We can specify a timepoint, such as t_0 + 3 minutes, whose origin (t_0) is the beginning of a presentation, called *relative time*. We may also specify a timepoint -- 12:45pm -- with respect to *world time* (perhaps Greenwich Mean Time). Of course, synchronizing to a specific world time instant will only have an effect if a process is executing the temporal specification when this instant occurs.

2.2 Temporal Specification

A temporal specification defines temporal relationships between temporal objects. These relations can be classified into two categories: *Object synchronization* and *Fine-grain synchronization*. Object synchronization, as previously described, is used to define the relative temporal ordering for the presentation of media objects. Fine-grain synchronization is used to indicate that a semantically stronger relationship exists between intervals being presented simultaneously. For example, we

can define a temporal relation indicating that two audio clips should start and end simultaneously. However, this relation suggests nothing about how the audio clips should be played in relation to each other. Due to the limitations of current systems (network delays, resource contention, loss of data, etc.) the playout rate of any media object will generally fluctuate. If these audio clips comprise left and right channels of a high-fidelity recording, the delay between the playout of related samples of each audio clip should be less than approximately 50ms (dictated by human perception of audible delays). Thus, fine-grain synchronization should be defined on these intervals to indicate there is a degree of synchronization that should be enforced by the system/application when these objects are played out.

Some models may define fine-grain synchronization as simply an extentional representation of an object synchronization relationship. That is, if we view each media object as being composed of smaller objects (e.g. atoms), we can simply define temporal relationships between the atoms at this level, using object synchronization. In actuality, most of these models support a macro-type object synchronization specification between media objects. This would result in relationships being defined between the atoms of the media objects. Since not all models view fine-grain synchronization in this way (extentionally), we have defined the two types of temporal relations as described above.

2.2.1 Object Synchronization

We summarize the expressiveness of a model's temporal relations using five descriptors: Basic relations, derivable starts, delayed starts and finishes, causality, and deferment.

2.2.1.1 Basic Relations

Basic relations describe a model's capability to represent very general temporal relations between intervals or endpoints. When discussing the expressive power of temporal models, it has been

customary to reference Allen's work on interval relations[18]. Allen has identified 13 relations that can exist between two intervals. However, these relations are between intervals; and, while some of the models use interval relations, most use endpoint relations. We define the following three *basic* relations (figure 1) that can be expressed using intervals or endpoints: START TOGETHER, START AT FINISH, and FINISH TOGETHER. FINISH TOGETHER is required to support unpredictable durations. Otherwise, use START TOGETHER with synchronization delays (discussed below) to achieve the same relation with intervals having predictable durations.

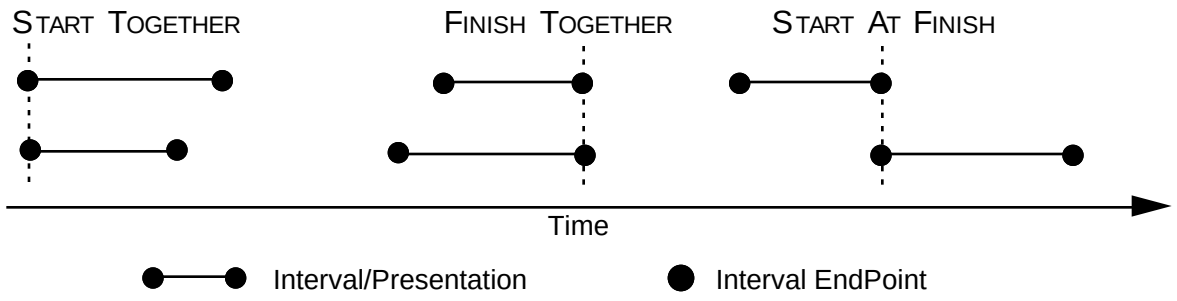

Figure 1.

If the model expresses (positive) delayed synchronization (defined below), it will be able to represent Allen's 13 relations.

2.2.1.2 Derivable Starts

Derivable starts permit a temporal specification that only involves specifying the interval's end point. So, for example, we may want to constrain a movie to end at 3:00 pm. We then leave it up to the system to determine a proper start time to satisfy this constraint. The interval's approximate start time can be derived by its ending time and predicted duration. Of course, deriving start times for intervals of unpredictable duration is not possible. Current timeline models define starting and ending times for each interval, and current hierarchic models use a top-down specification (relations involve the start of intervals). Therefore, neither can specify derivable starts.

2.2.1.3 Delayed Starts and Finishes

As described above, models should also be able to represent delayed starts and finishes, in order to extend the flexibility of a temporal specification. The basic relations allow precise alignment of interval endpoints. For example, interval I_A begins at the exact same time as interval I_B begins, or I_A begins exactly when I_B finishes. Supporting the specification of start and finish delays extends the expressiveness of a model. Thus, we can express such specifications as I_A starts 30 seconds after the end of I_B. Temporal specifications of any interval within timeline models are with respect to a timeline, rather than by specifying relations between intervals. Therefore, it is impossible to directly specify delayed starts and finishes within those models. However, the models can implicitly specify delays by assigning appropriate start and finish times to the intervals.

Models may specify delays that are *positive* values, *negative* values, or value *ranges*. The example above used a positive delay: "...start 30 seconds after the end... ." We cannot specify, however, that "a movie 'trailer' should end one minute **before** the feature movie presentation begins" using postive delays. To do so, we need to specify a negative delay (e.g. -1 minute). In this situation, the movie's start time must be known or derivable (either during compile time or runtime).

Ranges provide another dimension to specify delay specification. Range delays were discussed extensively in [2], and later in [23] as a means of supporting additional flexibility in specifying start and finish times. Providing an acceptable range of values shows that precise timing isn't always required, and it also provides the system (e.g. a runtime temporal formatter) with some flexibility in determining start or finish times, rather than requiring exact timing, which usually isn't possible anyway.

2.2.1.4 Causality

Causality enables the triggering of actions based upon the occurrence of events. Actions can take a number of forms, such as the starting, stopping, fast-forwarding, or rewinding of a media object.

Events can be basic, such as the begin event of a media object, or user-defined, such as defining a media-based event associated with the appearance of a red ball within a scene in a film. They can also be application-defined, such as a menu selection made by a user, or time events that are triggered when a timepoint associated with a specific clock occurs. (Remember that events can be classified as either affectable or unaffectable.)

In a basic way, we can define causality using two events X and Y, such that "the occurrence of event X causes the occurrence of event Y." How this specification is actually implemented varies from model to model. However, the essential behavior is that the occurrence of one event causes another event to occur.

As stated above, causality has an associated implementation at the system level. In general, most temporal specification models have defined the same basic behavior for causality, as described above. We say a model is flexible if the temporal specification of the model provides some alternatives as to its implemented behavior, rather than just one defined behavior.

2.2.1.5 Deferment

While causality brings about the occurrence of an event, deferment is used to inhibit an event from occurring. More specifically, we can inhibit (or defer) an event during a specified interval. We can define deferment using one event X and an interval i, such that : "event Y does not occur during interval i." Since time is always increasing, if event Y would have occurred during this interval, we can restate this as "event Y will not occur *at least* until the end of interval i." For instance, if a video is presented with background audio, a practical example of deferment might be to inhibit the end event of the audio *at least* until the video's interval ends. Note, however, that when the end of the video's interval occurs, it will not cause the occurrence of the end of the audio -- this must be specified using causality.

As with causality, how deferment is actually implemented is the behavioral aspect of the specification, and it varies from model to model. From the example used above, it is possible that the audio will end before the video (otherwise, the specification would not have been made). Some possible implementations include performing some other action such as repeating the audio , or decreasing the consumption rate so that the audio ends just when the video ends.

Deferment delays, like delayed starts/finishes, provide additional expressive power to a model (they are defined in the same way as delayed starts/finishes). Using delays we can define when the inhibition should cease relative to the occurrence of the inhibiting interval. If we include a deferment delay within a specification it should be read as follows: "event Y does not occur until *at least 2 minutes after* the time at which the end of interval i occurs." As above, models can provide greater *flexibility* by supporting more than one basic behavior at implementation.

2.2.2 Fine-grain Synchronization

When we require a finer degree of synchronization between intervals than just starting and stopping at the same time, we need to be able to express this requirement to the system. Fine-grain synchronization can be used to define synchronizing relationships between intervals that are playing out simultaneously. In general, fine-grain synchronization is required when fine timing relationships exist between the atoms that make up the intervals (e.g. video frames are the atoms of a video clip).

Referring to a prior example, the playing out of a movie and its related soundtrack requires a certain degree of synchronization between the video frames of the movie, and the audio samples of the soundtrack. This is required, for example, so that the actors in the film mouth words at the same time the words from the soundtrack are played over the speakers. The delay between the display of a video frame and the playing of the associated audio samples should be no more than

150ms, as previously discussed. Note that a delay of 0ms is not required, nor would it enhance the quality of the playout since average human perception requires "only" a delay tolerance of no more than 150ms. We can use this information to ensure the appropriate level of synchronization, without demanding more than is required.

In principle, fine-grain synchronization may be implemented solely by defining individual temporal relationships between the atoms of the intervals to be synchronized. Considering that high quality audio is recorded at 33,000 samples per second, defining these relationships between two one-hour audio recordings would be a daunting task. In actuality, most current models support the specification of fine-grain synchronization at the interval level. This is understood by the system to be a semantically stronger relationship, requiring a fine degree of synchronization between the atoms of the intervals.

A number of different approaches exist within current models for fine-grain synchronization. Some models provide only one level of synchronization; while others provide *variable* levels of synchronization -- for example, 0.0 could indicate no fine-grain synchronization is required, while 1.0 would indicate the finest level of fine-grain synchronization is required[6]. Some models treat fine-grain synchronization as an *extentional* relationship specified on intervals[2, 3, 6]. In turn, temporal relationships are actually specified between the atoms of the related intervals. For example, synchronizing two thirty minute videos (at 33 frames per second) would require 59,400 relationships with models supporting macros and one level of synchronization, or variable synchronization equal to 1.0 [2, 3, 6].

Finally, models can offer *flexibility* by supporting alternative methods for implementing synchronization -- for example, allowing a user to request a specific means of synchronization -- as the behavioral implementation [24].

3.0 Temporal Assembly Language

In this section, we will describe our temporal specification model and language. Our model includes three powerful and yet simple relations expressing causality, deferment and synchrony (fine-grain synchronization). Using these we have been able to express the relations that may be specified in all of the other models examined (see table 1). In addition we can express several relations that cannot be expressed in any of those models.

To assist in the understanding of this model we will define the following temporal specification scenario, which we will refer to throughout section 3 (figure 2). We have chosen to use a (slightly modified) graphical notation introduced in [19], for this example. Using this notation, the timeline has been turned on its side, where each media object has its own timeline (its length is proportional to its optimum duration). Square nodes represent media begin and end events, while round nodes represent system or user-defined events. The dashed lines represent media objects with unpredictable durations; and, arcs represent the temporal constraints that have been specified. Note that media objects are not positioned accurately in space relative to each other -- this is left to a temporal formatter, when it checks the validity of the specification.

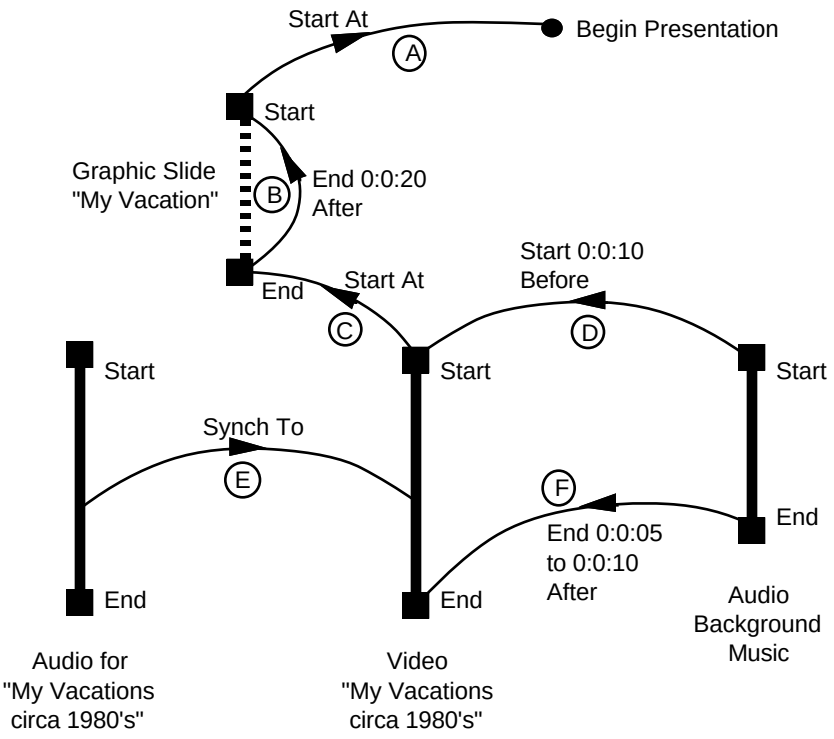

Figure 2. Example Temporal Specification

This example contains four media objects: two audio streams, one video and one graphic slide. The temporal relations of this example, marked by A thru F and indicated within circles, are described by:

A. The user begins the presentation by selecting a menu option "Begin Presentation", which should cause the graphic slide to be displayed.

B. The slide is displayed for 20 seconds.

C. The movie is then started when the slide is removed from the display.

D. Ten seconds before the movie begins, the background audio starts.

E. When the movie is presented, the audio track for the movie is also played requiring fine-grain synchronization (the movie and audio track start and end simultaneously).

F . Finally, after the movie ends, the background audio is played for an additional 5 to 10 seconds to "wind down" the presentation.

The notational conventions used within this model are as follows. The basic units are:

- Timepoints, T, which represent single instants in time. A timepoint is defined in relation to a user or system-defined clock, CLK_c. Two pre-defined clocks include: World time (such as Greenwich Mean Time) CLK_{World} and presentation-relative CLK_{P-REL}. At the beginning of a presentation the current time of the presentation-relative clock, CLK_{P-REL} is zero.

- Basic interval, t, which represents a duration with start point t_i and end point t_j, where $0 \le t_i \le t_j$; and,

- Delay interval, d, having a lower bound d_i and an upper bound d_j, where $d_i \le d_j$.

- Media interval, $m_{objectI}$, where $objectI$ is a media object, is a special class of basic interval. Each media interval has a startpoint, endpoint, and duration (determined by its execution). Media intervals are associated with the runtime execution (playback, recording) of media objects. Non-continuous media (e.g. text, or images) have no inherent duration. We define their start and end points respectively as the time to start and the time to end the presentation of the media object. As an example, the point at which an image is first displayed on a monitor is its associated start point, and the point when it is removed is its end point. The duration is derived from this time interval.

The model we describe is event-based, and defines three basic event types:

- A generic (including system, application or user-defined) event, EVT_i;

- Beginning and ending events of an interval t, $BEG(t)$ and $END(t)$, belonging to a special class of event; and,

- Time events, $TEVT(T)$, are a second special class of event . $TEVT()$ takes a timepoint T, and generates an event when time T occurs for clock CLK_c. (Remember, each T is associated with a specific CLK_c).

In addition, we define two time-related functions:

- occTime(EVT_i, CLK_c), which takes an event EVT_i, and a clock CLK_c,and returns a timepoint T defined in relation to clock CLK_c. It is defined as the occurrence time of an event. The default clock, if one is not specified is CLK_{P-REL}.

- curr_time(CLK_c) returns the current timepoint for the user or system-defined clock, CLK_c. As stated above the current time of the presentation-relative clock just before the start of a presentation is zero: curr_time(CLK_{P-REL}) = 0.

  We can also more precisely define the time at which a time event, $TEVT(T)$, occurs. This happens when:

$$\text{occTime}(TEVT(T), CLK_c) = \text{curr\_time}(CLK_c).$$

Each of the predicates we have defined include two types of specifications: the conceptual specification, and the behavioral specification. The conceptual specification facilitates the construction of temporal relations between events, while the behavioral specification is used to

define the behavior of the conceptual specification at implementation. We provide this distinction to separate the specification from its implementation. Rather than define specific implementations for each predicate, we have augmented the predicates to allow the user to select from an extensible set of behaviors.

3.1 Causality

The first predicate, **causes**, is used in a temporal specification to define causal relations. It takes two events, EVT_i and EVT_j, an optional delay interval, d, and optional system-defined extensions, SDExt:

causes(EVT_i, EVT_j, [d], [SDExt]).

A system-defined extension indicates a behavior for the specification. Rather than encode one behavior as other systems have done, we believe that a system should provide a flexible means of adding additional behaviors. Therefore, the user may select a non-conflicting set of these for each specification. Particularly for **causes**, we have defined only one behavioral specification. Others are discussed later on.

When the delay interval d = {0,0} (the default value for d), the **causes** predicate specifies the following: "the occurrence of event EVT_i will cause the occurrence of event EVT_j". Using the defaults for d and SDExt we have:

causes(EVT_i, EVT_j).

In the example of figure 2, there are two temporal relations where the delay interval is {0,0} -- described by statements **A** and **C**. In **A**, the two events are $EVT_{Begin-pres.}$ and $BEG(m_{MySlide})$. When the system generates the event $EVT_{Begin-pres.}$, it should cause the occurrence of the presentation of the slide. Therefore, we define the specification as:

$$\textbf{causes}(\ EVT_{Begin-pres.}, BEG(m_{MySlide})).$$

In **C**, the events are $END(m_{MySlide})$ and $BEG(m_{MyVideo})$. Here, we would like the end of the presentation of the slide (the occurrence of its end event) to cause the start of the video (the occurrence of its begin event). The specification should look like:

$$\textbf{causes}(\ END(m_{MySlide}), BEG(m_{MyVideo})).$$

Within a **causes** statement, the delay interval d is defined as the range $\{d_i, d_j\}$. The range defines an interval in which the event *must* be caused to occur. The meaning is: "the event EVT_j should occur sometime within this range (system-dependent[5]), and therefore, its occurrence time will then lie within the following time interval:

$$occTime(EVT_i) + d_i \leq occTime(EVT_j) \leq\ occTime(EVT_i) + d_j.$$

Note that events and timepoints are not of the same type and can not be added together directly; we determine the timepoint of an event, EVT_i using *occTime*(). Then, add the minimum (and maximum) value in the delay interval to determine the valid time interval. Any timepoint inclusively within this time interval is a valid point to cause EVT_j to occur (a system-defined

[5]One system-defined extension could provide functions that allow the user to define preferences over the delay interval. This would assign higher preferences to specific points or sub-intervals within the entire interval. As a default, all timepoints within the interval are assigned preference. The system uses this information in determining what point in the interval will be selected.

extension may define a weight/cost function across the interval, so that some points in the interval are more likely to be chosen than others). Thus, say the system selects a timepoint k such that:

$$d_i \le k \le d_j$$

and, therefore we have:

$$occTime(EVT_i) + d_i \le occTime(EVT_i) + k \le occTime(EVT_i) + d_j.$$

The initial specification can be restated as:

$$\textbf{causes}(\ TEVT(occTime(EVT_i) + k),\ EVT_j).$$

The delay interval in our model can specify positive and negative values, and range delays. No other models supports specifications using negative delays. We believe that negative delays are necessary to enable specification of natural temporal relations, as demonstrated in the example (statement **D**). Note, that as previously described, negative delays can only be used in conjunction with events whose occurrence time is known or predictable.

In the example, three temporal relations include delays; they are described by statements **B**, **D**, and **F.** In **B**, the two events are $BEG(m_{MySlide})$, and $END(m_{MySlide})$. The relation specifies that the end event of the slide should be caused by the begin event of the slide + 20 seconds. Therefore, the specification is simply:

$$\textbf{causes}(\ BEG(m_{MySlide}),\ END(m_{MySlide}),\ \{20,20\}).$$

Since the lower and upper values of the range are equal, the range specified simplifies to a single timepoint, defined by $occTime(END(m_{MySlide})) + 20$.

Since, statement **D** uses negative delays, we will first discuss **F**. In statement **F**, the relation indicates that the occurrence of the end event of the video plus 5 to 10 seconds should cause the end event of the music. The events involved are $END(m_{MyVideo})$, and $END(m_{MyMusic})$. We can define the specification as:

$$\textbf{causes}(\ END(m_{MyVideo}),\ END(m_{MyMusic}),\ \{5,10\}).$$

When negative delays are used, the timepoint at which the associated event occurs must either be known in advance (such as a time event), or the system must be able to estimate it; otherwise, the specification using a negative delay would be invalid.

In statement **D**, the relation specifies that the occurrence of the begin event of the background music should occur 10 seconds *before* the occurrence of the begin event of the video. From the specification we have developed, the begin event of the video will occur (indirectly) 20 seconds after the begin event of the slide. The two events in this specification are: $BEG(m_{MyVideo})$, and $BEG(m_{MyMusic})$; and the resulting specification is:

$$\textbf{causes}(\ BEG(m_{MyVideo}),\ BEG(m_{MyMusic}),\ \{\text{-}10,\text{-}10\}).$$

With respect to its implementation, using a runtime temporal formatter like that described by Buchanan and Zellweger [19], we believe that it would be possible to implement a capability to approximate the begin event of the video by approximating the end event of the slide. (Remember, that the begin event of the video is caused by the end event of the slide.) As the end event of the slide approaches, the following timepoint will also approach $occTime(END(m_{MySlide})) - 10\,\text{sec}$ (end event of slide minus 10 seconds) or equivalently, $occTime(BEG(m_{MyVideo})) - 10\,\text{sec}$ (begin event of

the video minus 10 seconds). When this timepoint occurs, the system should cause the occurrence of the begin event of the music.

The resulting specification is evaluated as:

$$\textbf{causes}(\, TEVT(occTime(BEG(m_{myVideo})) - 10\,\text{sec} + \Delta t),\ BEG(m_{myMusic}))$$

where Δt is ideally as close to zero as possible. In general, if the triggering event (such as the begin event of the video minus 10 seconds) is itself dependent upon other events, its occurrence time can only be estimated. There are two reasons for this: 1) dependent events may be canceled by the user; or, 2) the actual timepoint at which a dependent event occurs may be different than the specification due to presentation delays (such as loss of data). Therefore, the runtime formatter must continuously update the estimated value for the event; in our case:

$$occTime(END(m_{mySlide}) - 10).$$

Updating can occur until some timepoint $\Delta t - 10\,\text{sec.}$ before the estimated end event of the slide. Optimally, Δt will be close to zero. Note, however, that the intention of this paper is to discuss how this model provides the ability to define temporal relations, and not to define a precise system-level implementation.

With any temporal specification that is based upon the assumption of an event's occurrence, it is possible that an "incorrect" execution will occur. For example, in the simple specification "*start the audio 3 minutes before the feature presentation*", the audio may be started and the feature presentation may not start due to a mechanical failure of the projection system. Certainly, some events may be more likely to occur than others, so perhaps confidence levels could be assigned to these events. A specification may then refer to a confidence level, such as "*start the audio 3*

minutes before the feature presentation, if there is at least a 90% chance that the feature presentation will start on time."

Clarification of two aspects of **causes** must be addressed; the first pertains to derivable starts. The following **causes** statement specifies that "the occurrence of the beginning of a movie will cause the occurrence of the end of the audio."

$$\textbf{causes}(\ BEG(m_{movie}),\ END(m_{audio})).$$

Note that it does not specify when the audio should begin. If the beginning event of the audio interval is not part of another specification (and therefore the timepoint at which the audio will begin has neither been directly or indirectly specified otherwise), this statement may be used to determine the start time of the audio (e.g., derived start). The system could derive a specification for the start as (the event equal to the occurrence time of the beginning of the movie minus the predicted duration of the audio will cause the occurrence of the begin event of the audio).

$$\textbf{causes}(\text{TEVT}(\text{occTime}(\ BEG(m_{movie})) - \text{length}(\text{audio})),\ BEG(m_{audio})).$$

The second aspect of **causes** we must clarify is the types of relations that can actually be specified. If we restrict our temporal specification to two media objects and their associated events (BEG and END), we can specify four basic relations:

- Begin-at-Begin -- Begin one interval at the beginning of another interval
- Begin-at-End -- Begin one interval at the end of another interval
- End-at-End -- End one interval at the end of another interval
- End-at-Begin -- End one interval at the beginning of another interval.

Again, if we limit the delays to positive and negative delays (no ranges), each relationship has three possible forms. Using one relation, Begin-at-Begin, and adding delays it becomes Begin-at-Begin + t where t>0, t=0 or t<0. The three forms are then:

- Begin-at-Begin -- Begin one interval at the beginning of another interval (t=0)
- Begin-at-Begin + t -- Begin one interval t units *after* the beginning of another interval
- Begin-at-Begin - t -- Begin one interval t units *before* the beginning of another interval.

If we specify temporal relations using only one of the events from each of the two media objects, we can specify 12 relations (four basic relations x three forms each). If we specify relations on these media objects that involve both their BEG and END events, there are an additional 36 possible relations that can be expressed for a total of 48 relations -- that only involve two media objects. If we can use any temporal objects (e.g., time events, user-defined and application events, finer-grain events), and include range delays, the number of relations that just the **causes** predicate can express is quite large.

Just as an example of contrast, current hierarchic models can express two of these relations: start-at-start (parallel) and start-at-end (sequential), where t=0, or t>0. These models can simulate the third basic relation end-at-end, finish together, using the start-at-start relation and null objects (their form of delayed synchronization) [17]. Null objects delay the beginning of each media object so that their ending times will occur simultaneously (the length of each media object must be known). Still, current hierarchic models can only express Allen's 13 relations (they include no other temporal objects to synchronize to).

3.2 Deferment

The second predicate, **defers**, is also used within a temporal specification to define temporal relations. However, **defers** is used to *inhibit* events using deferment. The notion of deferment compliments the notion of causation: using two predicates we can cause an event to occur, and inhibit an event from occurring. **Defers** takes one event, EVT_i, an interval t, an optional delay value d_i, (default $d_i = 0$), and an optional set of system-defined extensions SDExt (as described above). The interval t is either a basic or media interval, composed of bounding events $BEG(t)$ and $END(t)$, where by definition,

$$occTime(BEG(t)) \leq occTime(END(t)).$$

The form of the statement is:

defers(t, EVT_i, [d_i], [SDExt]).

Again, if we use the defaults, the specification simplifies to:

defers(t, EVT_i).

This specification should be interpreted as: "interval t defers EVT_i"; or, not quite so tersely as: "if EVT_i would occur during interval t, the occurrence of event EVT_i will be deferred <u>at least</u> until the occurrence of the end event of interval t."

The deferment of an event can have several implementational behaviors. For some events, it may be desirable, if possible, to actually disable the causing factor of the event (such as disabling a menu option). While for other events, we may only be able to defer the time at which the event is

actually registered. At the implementation level, within an object-oriented environment, objects may define their own methods to handle a defer-related system extension invocation.

Therefore, when an event is deferred, the cause of the event may or may not be disabled -- it is a behavioral implementation choice. As an example, we may construct a graphical menu with a "Quit" option that generates an event, EVT_{QUIT}, when selected. If we defer this event during a specified interval, two of the possible implementations may include: disabling the menu option "Quit" so it cannot be selected (this disables the cause of the event) during the deferment interval; or, buffer EVT_{QUIT} if the menu option "Quit" is selected, and then register it once the end of the interval occurs. The event, EVT_{QUIT}, described above is an example of an *interval-independent* event. Interval-independent events are not associated with the beginning or ending of a basic interval.

Remember, that basic intervals have begin and end events, where the duration of an interval is defined as:

$$duration(t) = occTime(END(t)) - occTime(BEG(t))$$

Events such as these are called *interval-dependent* events since they affect the duration of intervals. For example, the deferment of an end event can modify the duration of an interval; by deferring the end event of an interval, the duration of the interval will be longer to the extent that the end event was deferred. (Note that the deferment of a begin event will not alter the duration of an interval, since only the start time is modified.) Therefore, **defers** can increase the duration of an interval. For example, we can cause the begin event of a two hour movie to occur at 1:00pm, and defer its end event until at least after the end of the interval {1:00pm, 4:00pm} -- the system must defer the end event from occurring until at least this time. The behavior we select for this specification at implementation may be: to play the movie slower so that it lasts for at least 3 hours (until at least

4:00pm); to restart the movie at 3:00pm (when it would otherwise end, since it is two hours in length) to play the movie again; or, perhaps some perform some other action (similar to alternate actions in [6]) offered by another system-defined extension.

In contrast, **causes** can shorten the duration of an interval by causing the end event to occur earlier than it would occur naturally. For example, if we cause the begin event of a two hour movie to occur at 1:00pm, and cause its end event at 2:00pm, we have shortened the duration of this interval to one hour. Implementations of this specification include speeding up the rate of playback so that the movie can be played in one hour; or, truncating the movie when 2:00pm occurs (other system-defined implementations may also be supported). We leave it to a temporal formatter to identify intervals that have been lengthened by a deferment, or shortened by a causal specification.

As with **causes**, we can also specify positive and negative delays. We haven't found range delays to be particularly useful[6] , so currently we do not support range delays for this predicate.

With respect to our example in figure 2, statement **F** requires a deferment relation to be specified. Why? The background music interval is shorter than the movie, and it shouldn't end until at least 10 seconds after the end of the movie. For the deferment specification, the event is $END(m_{MyMusic})$; the deferment interval will be $m_{MyVideo}$; and, the delay will be 5 seconds (we want to defer the end of the music until at least 5 seconds after the video ends). We will also assume for this example, that a system-defined extension exists, "Expand", that will slow the music's rate of play so it ends at least 5 seconds after the movie. Therefore, the specification will be:

$$\textbf{defers}(m_{MyVideo}, END(m_{MyMusic}),5, \text{“Expand”}).$$

[6]The range delay for a **cause** statement defines a closed interval. The event should not occur before d_i not later than d_j. For a deferment, the interval is open. That is, the event should not occur until at least d_i. Specifying an upper bound on the interval has no meaning when the interval is open-ended (at some future time their may be some motivation to attach some meaning to this).

We can interpret the above example as, "the end event of the music will be deferred until at least 5 seconds after the occurrence of the end event of the video."

Using the above specification, if the end of the interval $m_{MyMusic}$ associated with the music will naturally occur after the end interval $m_{MyVideo}$ associated with the video, this specification will have no resulting effect. If the music will end before the video, this specification will have some effect dictated by the behavioral implementation chosen by the user as described above.

The deferment specification delays the end of the music so that it won't occur *at least until* the desired time. If we want to ensure that the end event for the music does occur at a particular time, we need a causal specification. In the example of figure 2, the temporal relation also indicates that we want the music to end 5 to 10 seconds after the video. Thus, within the temporal specification for this presentation, we need to define three statements related to the presentation of the music. We have previously defined the following statements for this purpose:

To start the music, we defined:

$$\textbf{causes}(\ BEG(m_{MyVideo}),\ BEG(m_{MyMusic}),\ \{-10,-10\}),$$

and, to defer the end of the music, we defined:

$$\textbf{defers}(\ m_{MyVideo},\ END(m_{MyMusic}),5,\ \text{"Expand"}),$$

and, to end the music, we defined:

:

$$\textbf{causes}(\ END(m_{MyVideo}),\ END(m_{MyMusic}),\ \{5,10\}).$$

3.3 Fine grain synchronization

Fine grain synchronization in our model is specified using the primitive **synch**. It takes two intervals t_m, t_n, where t_m and t_n may or may not have the same length; a synchronization factor, synch$f$, to support variable level synchronization; and, an optional set of system-defined extensions, SDExt, for the behavioral specification.

The intervals are either basic or media intervals, t , composed of bounding events $BEG(t)$ and $END(t)$, where by definition,

$$occTime(BEG(t)) \leq occTime(END(t)).$$

The **synchs** specification has the form:

$$\textbf{synchs}(t_m,\ t_n,\ \text{synch}f).$$

This should be read as, "synchronize the presentation of the atoms of t_n to the atoms of t_m, at least to the degree specified by synch$f$." (As stated above, these intervals may have different lengths. What actually occurs at implementation can be specified using system-defined extensions - see section 4 for a discussion of some extensions.)

In some models [6], variable synchronization is specified by setting a synchronization value within the range from 0.0 to 1.0, as described in section 2.2.2. While synch$f$ = 1.0 is allowable, in general it is probably not achievable. More importantly, for most applications it is not required. As discussed in section 2.2.2, due to the limits of (average) human perception, delays between the display of related atoms from each media interval are tolerated to seemingly well-defined values.

However, we should clarify that a single synchronization value can only be used as a high level indicator. It should be treated as a first-order specification for the enforcement of synchronization between the atoms of the intervals. At an implementation level, additional constraints would need to be specified such as the maximum number of frames that can be lost within an interval (loss rate), and the maximum number of frames that can be lost in a row (bursty loss rate).

It is also our opinion that variable synchronization should be supported within a temporal specification. This allows the system to determine what degree of synchronization it will provide, bounded by perfect synchronization and the synchronization factor specified by the user. How 'perfect synchronization' is defined and enforced is system-dependent. However, within our model, we support enumerated values for synch$f$, rather than real values in the range 0.0 to 1.0. An example of enumerated values might include: "High Fidelity Audio", "High Definition Audio/Video", "Standard Audio", "Standard Audio/Video", etc... Of course, definitions for these values must be supported by the system. Thus, if two CD-quality symphony tracks were to be synchronized, we may set synch$f$ equal to "High Fidelity Audio". These values can carry additional meaning that may help to define second or third-order constraints at the implementation level that is generally not possible with the first approach.

For the last temporal specification required in our example of figure 2, we need fine-grain synchronization between the video and audio to ensure that we do not have a lip-sync problem. The two intervals then are: $m_{MyVideo}$ and $m_{MyAudio}$. synch$f$ will be "Standard Audio/Video" since the audio and video were recorded using a video camera that didn't support high definition audio/video -- we will presume that "Standard Audio/Video" will satisfy our requirements. This specification will then be:

synchs($m_{MyVideo}$, $m_{MyAudio}$, "Standard Audio/Video").

Since our model separates conceptual specification from behavioral implementation, it can support any number of implementations. For fine-grain synchronization we offer a brief description of one possible implementation for synchronization based upon rate drift constraint. By constraining the rate drift between the atoms within the intervals we can ensure synchronization. We begin with a simple example of two media intervals, m_{musicA} and m_{musicB}, having the same duration and same number of samples (atoms). We can define the rate drift, r, for the playback of sample i of each media interval as:

$$r = \left| Sample_{i.musicA}.StartTime - Sample_{i.musicB}.StartTime \right|.$$

To constrain the rate drift throughout the interval, we can specify a constraint interval c, such that:

$$\forall(i)\left(\left|Sample_{i.musicA}.StartTime - Sample_{i.musicB}.StartTime \right| \leq c\right).$$

For high fidelity stereo, we could constrain c = 50 ms. If the media intervals were of different types, or the ratio of atoms isn't 1:1, a slight variation of this is required.

Other approaches, such as timestamping (e.g. [24]) or synchronization points (e.g. [6], [2], [3]) require adding synchronization information to the media itself. Unless this information can easily be modified at runtime, changes to the rate of playback or quality of service will be problematic for these approaches.

3.4 Additional Specification

In addition to the temporal relations that can be specified in this model, other information may be required for the playback of media objects. Two examples include quality of service (QoS), and the consumption rate σ. Each of these are associated with the presentation of the individual media objects. Again, σ, is the rate at which the atoms of a media object are displayed. This value may be directly modified, say to twice the nominal rate, 2σ. It may also be altered indirectly by a system-defined extension associated with a **defers** or **synchs** specification, that may have compressed or expanded its associated interval (accomplished by modifying the rate).

QoS can be specified at the media object level, or on entire multimedia presentations. To buffer the implementation details from the specification we suggest that QoS levels are used, perhaps in the form of enumerated types. They would range from a high quality of service to a low quality of service, similar to that used by the synch$f$ variable in the **synchs** predicate.

4.0 System-defined extensions

In this section we present some system-defined extension examples for the **causes** and **synchs** predicates. When a temporal relation is specified using **causes**, one possible extension could allow the user to specify whether the synchronization should be *relative* or *absolute*. Using an example, say the end of an audio causes the beginning of a video. With relative synchronization, the beginning of the video will occur when the end of the audio actually occurs. While using absolute synchronization, the beginning of the video would occur when the audio would end, barring any delays, such as network delays.

In addition, we have also considered extensions for **synchs** to handle runtime relationships and the synchronization of uneven intervals. First, when specifying fine-grain synchronization for runtime interactions, it may be useful to specify additional semantics on this relationship. These relationships may define dependencies, for example, that affect the presentation (similar to [3]). Two of these include dependent, and co-dependent. A dependent relationship between synchronized presentations indicates that the playout of one presentation (*A*) will be dependent upon the state of the other presentation (*B*). For example, if *A* is stopped, *B* will also be stopped, etc.

Since two presentations that are synchronized must have the same runtime duration (in our model), when unequal length presentations are synchronized we think that the user should be able to specify, through an extension, how to equalize the durations of these presentations. The user can either shorten the longer presentation, or stretch the shorter presentation. To shorten the longer presentation, the user may (for example) truncate it at its end so that both presentations are of equal length; or, the user can compress one presentation by increasing the rate associated with the playout. To stretch a presentation, it can either be expanded by decreasing its rate; or, perhaps some pre-defined alternate action can be specified similar to that used in the restricted-blocking of Steinmetz [2].

5.0 Conclusions/Summary

In this paper we have presented a new event-based temporal specification model using three simple relations expressing causality, deferment and fine-grain synchronization. Using the powerful primitives of our language, **causes**, **defers**, and **synchs** we have been able to express the temporal relations of all models we have been able to study. In addition, there are many relations our model can express that cannot be expressed by any of these models -- for example, causality

with negative delays, and deferment with positive or negative delays. Our approach to distinguish between the specification of temporal relations and their behavior at implementation has resulted in a very flexible and extensible language.

Although our model is quite expressive, we have refrained from adding additional capabilities for two reasons: 1) this would have the affect of increasing the complexity of the language -- we strongly feel that simplicity is very important from the users' perspective; 2) it is not our intention to define a temporal model that facilitates specification of every conceivable temporal relationship. We do believe that embedding our predicates within a language such as C++ will provide additional capabilities that are currently not supported.

In addition, although we haven't discussed support for alternate presentations that may be based upon resource and hardware availability for example, we do believe that the use of system-defined extensions should support this. For example, one extension may indicate that a specification is conditional, based upon some specific resource availability (we view this as a behavioral implementation rather than a conceptual specification).

We have not yet examined the affects of playing temporal specifications in reverse, which have been discussed in [3, 16]. Nor have we addressed spatial specification (when and where windows should be displayed for example), which have been discussed in [3], and implemented in [6].

Future Directions

It is expected that an implementation of this model will be incorporated into a project similar to Gibbs' programming environment project which uses a data flow model [25] (current data flow models do not support the specification of temporal relations.) Secondly, we would like to build or extend a temporal formatter to support our model.

We also anticipate future modifications to the model during implementation, and probably after some use. However, the three basic relations do allow the specification of more temporal relations than the models we have examined. Although, this is a good indication of the comparative expressibility of our model, it doesn't provide any insight into the ease of use within a real multimedia presentation development environment.

References

1. Blair, G., *et al.*, *An Integrated Platform and Computational Model for Open Distributed Multimedia Applications.* 1992. : p. 223-236.

2. Steinmetz, R., *Synchronization Properties in Multimedia Systems.* IEEE Journal on Selected Areas in Communications, 1990. **8**(3): p. 401-412.

3. Schnepf, J., D. Du, and J. Liu, *Synchronization for an Interactive Multimedia Presentation System*. 1993, University of Minnesota.

4. Gibbs, S., *et al*. *A programming environment for multimedia applications*. in *2nd Int'l Workshop on Network and Operating system support for Digital Audio and Video*. 1991. Heidelberg, Germany.

5. Blakowski, G., J. Huebel, and U. Langrehr. *Tools for specifying and executing synchronized Multimedia presentations*. in *2nd Int'l Workshop on Network and Operating system support for Digital Audio and Video*. 1991. Heidelberg, Germany.

6. Blakowski, G., *Tool Support for the Synchronization and Presentation of Distributed Multimedia.* Computer Communications, 1992. **15**(10): p. 611-618.

7. Pazandak, P. and J. Srivastava, *A Multimedia Temporal Specification Model Framework and Survey*. 1994, University of Minnesota. Technical Report in progress.

8. Vazirgiannis, M. and C. Mourlas, *An Object Oriented Model for Interactive Multimedia Presentations.* The Computer Journal, 1993. **36**(1): p. 78-86.

9. Mano, *Object Model Facilities for Multimedia Data Types*. 1990, GTE.

10. Gupta, A., T.E. Weymouth, and R. Jain. *An Extended Object-Oriented Data Model For Large Image Bases*. in *SSD 1991*. 1991. Zurich.

11. Ishikawa, H. and e. al., *The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System.* ACM TODS, 1993. **18**(March): p. 1-50.

12. Gibbs, S., C. Breiteneder, and D. Tsichritzis, ed. *Data Modeling of Time-Based Media*. Visual Objects, ed. D. Tsichritzis. 1993, Centre Universitaire D'Informatique: Université De Genève.

13. Drapeau, G.D. and H. Greenfield. *MAEstro - A Distributed Multimedia Authoring Environment*. in *USENIX*. 1991. Nashville, TN.

14. Gibbs, S., C. Breiteneder, and D. Tsichritzis, *Audio/Video Databases: An Object Oriented Approach.* IEEE Proc Data Engineering, 1993.

15. Hamakawa, R., H. Sakagami, and J. Rekimoto. *Audio and Video Extensions to Graphical Interface Toolkits*. in *Network and Operating Systems Support for Digital Audio and Video. Third Int'l Workshop Proceedings*. 1992. Germany.

16. Little, T.D.C. and A. Ghafoor, *Interval-based Conceptual Models for Time Dependent Multimedia Data.* IEEE Trans. on Knowledge and Data Engineering, 1993. **5**(4): p. 551-563.

17. Wijesekera, D., D. Kenchamanna-Hosekote, and J. Srivastava, *Specification, Verification and Translation of Multimedia Compositions*. 1993, TR94-1, University of Minnesota.

18. Allen, J.F., *Maintaining Knowledge about temporal intervals.* Communications of the ACM, 1983. **26**(11).

19. Buchanan, M.C. and P.T. Zellweger. *Automatic Temporal Layout Mechanisms*. in *ACM Multimedia 93*. 1993. California: ACM.

20. Fujikawa, K., *et al*. *Multimedia Presentation System "Harmony" with Temporal and Active Media*. in *USENIX*. 1991. Nashville, TN.

21. Horn, F. and J.B. Stefani, *On Programming and Supporting Multimedia Object Synchronization.* The Computer Journal, 1993. **36**(1): p. 4-18.

22. Esch, J.W. and T.E. Nagle. *Representing Temporal Intervals Using Conceptual Graphs*. in *Proc. 5th Annual Workshop on Conceptual Structures*. 1990.

23. Buchanan, C.M. and P.T. Zellweger. *Scheduling Multimedia Documents Using Temporal Constraints*. in *Network and Operating Systems Support for Digital Audio and Video. Third Int'l Workshop Proceedings*. 1992. Germany.

24. Gibbs, S. *Composite Multimedia and active objects*. in *OOPSLA*. 1991.

25. Gibbs, S. *Application Construction and Component Design in an Object-oriented Multimedia Framework.* in *Network and Operating Systems Support for Digital Audio and Video. Third Int'l Workshop Proceedings*. 1992. Germany.

Temporal Framework for Comparisions of Media Synchronization Models	Interval Specif.						End Point Specification										
	Timeline			Hier-archic		CI	Œ	Sync Point			Event Language			Events		
	Drapeau, Greenfield	Gibbs, Breiteneder	Hamakawa, Sakagami	Little, Ghafoor	Wijesekera, et. al.	Allen	Nagel, Esch	Steinmetz	Blakowski, Hübel	Schnepf, Du, Liu	Horn, Stefani	Vazirgiannis,Mourlas	Blair,Coulson	Buchanan, Zellweger	Fujikawa, et. al.	Pazandak, Srivastava
Temporal Objects																
Media																
Intervals	●	●	●	●	●	●										
Endpoints							●									
Media Durations																
- Predictable	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
- Unpredictable								●	●	●	●	●	●	●	●	●
Events																
- Begin/End								●	●	●	●	●	●	●	●	●
- Finer Grain								●	●	●	●	●	●	●	●	●
- Application											●	●	●	●	●	●
- Time											●					●
Timepoints																
- World Time																●
- Relative Time	●	●	●						●	●			●			●
Temporal Relations																
Object Synchronization																
Basic Relations	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Derivable Start	—	—	—	—	—	—				●				●		●
Delayed Starts/Fin.																
- Positive	—	—	—	●	●			●	●	●	●			●	●	●
- Negative	—	—	—													●
- Range								●					●	●		●
Causality										●	●	●	●	●	●	●
- Flexibility																●
Deferment			●[3]					●[3]	●[3]	●[3]				●[2]		●
- Delays																
- Positive																●
- Negative																●
- Range																
- Flexibility								●[3]	●[3]	●[3]				●[2]		●
Fine-Grain Synchronization																
- Unspecified	●		●	●		●	●				●	●		●	●	
- Supported		●			●			●	●	●			●			●
- Absolute					●			●		●						
- Variability		●							●[1]				●[2]			●
- Extensional					●			●	●	●						
- Flexibility		●														●

— Models can't directly express ●[3] Mix of deferment and causality ●[2] Unclear if actually supported ●[1] Restricted

Table 1. Framework & Comparisons