A Framework for Low Level Analysis and Synthesis to Support High Level Authoring of Multimedia Documents

Scott E. Hudson
Chen-Ning Hsi

Graphics Visualization and Usability Center
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

ABSTRACT

This paper describes a low level hierarchical framework for composition of multimedia documents that is designed to support the types of analysis, error checking, and synthesis techniques needed for rich user support in a high level authoring system. This framework supports flexible synchronization and flow control primitives as well as a range of interactive, continuous, and discrete media in a uniform fashion. The compositional approach taken in this work supports a style of analysis similar to the semantic analysis performed in programming language translation. This analysis capability provides a robust base upon which a number of potential high-level authoring facilities can be built.

Keywords: multimedia, composition, analysis, synthesis, synchronization, flow control, interactivity, authoring systems, continuous media, discrete media.

1. INTRODUCTION

The area of multimedia applications has been growing rapidly as new forms of interactive presentation media become accessible and affordable on various platforms. The use of new information channels, such as video, audio, and animation improves the expressive power of the computer as well as the attractiveness of applications. However, these richer sources of information also imply the need for powerful new tools for creating and controlling multimedia documents. New continuous media, such as audio and video, offer new challenges for authoring systems because they introduce a temporal aspect to document design. Dealing with the scheduling and synchronization issues that this implies (on top of the other normal artistic design, composition, and presentation issues) can greatly increase the difficulty of producing a successful multimedia document.

Techniques to support authoring of materials with a temporal component have primarily fallen into four categories: supporting time varying continuous media only as annotations to discrete (static) media, use of timeline notations, general concurrency specification techniques, and compositional approaches.

This work was supported in part by the National Science Foundation under grants IRI-9015407 and DCA-9214947.

The simplest form of support for time varying media is the use of continuous media annotations. to conventional (static) documents. This approach can be seen as a form of hypertext enhancement where the user "jumps" to the continuous media presentation then returns to the static text. Paradise, Metacard and Etherphone [Zell88] fall into this category.

Scheduling and synchronization issues under this approach are easy since control over these aspects of the continuous media are always placed directly in the user's hands (i.e., a single continuous media object *plays* at a time based on specific actions or requests by the user). However, this approach provides very little opportunity to combine different media in flexible ways and relies entirely on the user for pacing and control.

The second primary type of temporal specification is the use of a timeline. Under this approach, time is explicitly represented as a metered timeline similar to a musical staff. Media presentation objects are then laid down along the timeline to define their start, duration, and relative timing. The MAEstro system [Geor91], for example, falls into this category.

This approach allows both the start time and the end time of a media object to be specified and synchronized with the start and end times of other objects in a flexible and easy to understand manner. However, this approach requires that the exact duration and pacing of media objects be known in advance so that they may be placed on the fixed time line in a lock-step fashion. This makes it very awkward to deal with media containing user interaction or other unpredictable elements. Limited support for these media types typically can be provided only outside the timeline via commands that manipulate (e.g. explicitly stop or reposition) the clock controlling the overall timeline.

To overcome these limitations and provide a great deal of control over scheduling and synchronization issues, a third approach — use of general purpose concurrent programming notations — has been applied in several systems. These systems typically employ or adapt one of the high-level specification methods developed for concurrent programming (such as path expressions [Camp74, Alle83, Hoep91] or Petri-nets [Pete81, Stot89]).

These approaches support interactive and other unpredictable media cleanly and provide a great deal of expressive power. However, these systems — precisely because of their generality and power — are often difficult to understand and debug. In some sense they tend to be too powerful and generally too low level for direct use by authors. (Related to these techniques, but perhaps striking a better balance, are new approaches for specifying general synchronization through temporal constraints [Buch92a, Buch92b, Kumm91]).

In this work we attempt to find a middle ground between the flexibility and control of general concurrency specifications and the ease of use of timeline and annotation approaches. To do this we employ the fourth primary approach to temporal specification — the compositional approach.

Under the compositional approach (see for example [Fium87, Gibb91]), multimedia documents are constructed by composing small presentations into larger ones. Temporal relationships are created using special scheduling and synchronization composition operators. This approach allows techniques such as timelines or general synchronization notations to be applied, but only in a structured hierarchical setting. This restriction to a hierarchy can be limiting in some cases, since arbitrary temporal relationships across hierarchical boundaries cannot be expressed. However, this loss of power is not a problem for most documents in practice and, as we will demonstrate in this work, the hierarchical structure provides other advantages. In particular, in this work we will show how a hierarchical composition structure can be used to support sophisticated new analysis and synthesis techniques which can be used to provide very high-level authoring support.

Although as usable as other systems of the same type, we believe that the specific compositional framework described here is still probably too low-level to be effective for direct use in many authoring tasks. The intent of this framework will be to form a solid underlying basis upon which higher-level authoring facilities can be built rather than to provide the sole authoring interface in itself.

Because of this approach, emphasis has been placed on the generality and uniformity of the framework and on support for new techniques that can be used to provide analysis, error checking, and automatic synthesis of candidate solutions for detected problems.

Specifically, the framework presented here offers four central advantages. First, it allows interactive media — that is media whose action and timing is determined at least in part by the end-user — to be treated uniformly with other types of media. Second, it supports flexible flow control abstractions which will allow media of many different types to be scheduled and synchronized at a relatively high level and which will support explicit control by the end-user when desired. Third, by introducing a new abstraction for continuous media which separates the active control of timing from the more passive media specific presentation, it provides facilities for easy fine-grained scheduling, synchronization, and composition. Finally, and most importantly, it provides a framework for new error checking, analysis, and synthesis techniques that can provide very high level support in an authoring system.

An example of the kind of high-level authoring tools which we propose to support through analysis techniques might be an automated resource conflict detection and correction facility. This facility would use the analysis capabilities of the framework to detect and then isolate specific places where resource limitations could be exceeded in a presentation (e.g. points at which two audio tracks might need to be played on a single audio output device). Based on additional analysis of the context in which the problem occurs, the system might then be able to offer several ways to introduce additional specifications to solve the problem. For example, one audio presentation could be given unconditional priority over the other to resolve the conflict. Alternately, if analysis indicates that timing constraints will always temporally align the presentations, the material might be preprocessed by mixing it into a single presentation. Based on the analysis, the user could be allowed to either select a suggested solution, or create one of their own. If a suggested solution is utilized, the system could automatically synthesize the compositions and property settings necessary to carry it out.

A second example might be a synchronization constraint checking facility. This mechanism would automatically check to ensure scheduling feasibility — reporting for example that two presentations are required to start and end at the same time, but are of different lengths. The facility could then go on to suggest several ways to overcome this problem such as modifying the synchronization composition operator to truncate the slower presentation or wait until both are finished. Alternately, if analysis indicates that the media types involved support it (or if certain "wrapper" compositions could be applied), several ways of stretching or shrinking presentations could be suggested.

By providing a uniform framework for composition, analysis, and synthesis, we believe that a number of these very high-level authoring aids can be constructed. In the rest of this paper we will explore the framework and how it can be used to support these goals. The next section will consider a unified abstract representation that can be used to model a variety of media types. Section 3 will consider the common properties of this abstraction across all object types. Section 4 will then go on to describe a powerful set of composition operators, and Section 5 will consider their common properties. Section 6 will then describe an example multimedia document constructed with a prototype of our system and

Section 7 will go on to describe the analysis and synthesis techniques supported by the framework. Finally, Section 8 will provide a conclusion and discuss directions for future work.

2. A UNIFIED APPROACH TO VARYING MEDIA TYPES

In order to be able to perform interesting analyses on a variety of different multimedia object types, the framework described here employs an abstract representation which models objects with a range of different characteristics in a unified manner. This representation is used to support both time varying continuous media and static or discrete media. It is also used to model both primitive media objects and composed objects built up from other primitive or composed components.

In this abstract representation, each object is modeled as having four parts: A *content*, a set of *resource requirements*, a *timing engine*, and a *presentation driver*.

The content of an object is type dependent and contains the information that is actually to be presented (such as text, graphics, audio samples, or video frames).

The resource requirements of an object indicate what hardware or other resources it needs to create its presentation. These resource requirements are expressed in a *resource profile* which contains a list of resource needs and the time intervals associated with the (potential) need for that resource. For primitive media objects this profile will typically contain only one time interval (matching the duration of the object) for one resource. However, for composite objects the profile may be more complex.

The timing engine and presentation driver together control the actual delivery of an object's presentation. The timing engine is an active component that controls the timing of various parts of the presentation (the *when* of the presentation). The presentation driver is a passive component responsible for the actual manipulation and presentation of the object's contents based on timing triggers received from the timing engine (the *how* and *what* of the presentation). As will be indicated in Section 4, this splitting of responsibilities will enable more powerful compositions to be supported.

Note that some media types (for example, compressed video displays) may inherently internalize and combine the timing engine and presentation driver. However, in the system we will still model these aspects separately — typically providing a "fake" timing engine which logically corresponds to the actual engine embedded in the media hardware or software implementation of the presentation driver. This will allow compositions for fine-grained synchronization (as described in Section 4) to operate on these objects in a unified fashion.

Presentation drivers will typically be quite media type dependent. Some timing engines may also be media type dependent, but most will come from (or at least be modeled by) a library of reusable engines.

The simplest engine type is the *periodic timer* which simply fires at a set frequency. This would be used, for example, to drive or model video media with a fixed frame rate. Other timing engines may be more complex, producing timing triggers at varying intervals based on their own internal state. One of these later types of engines is the *cyclic timer* which repeatedly cycles through a list of scheduled triggers. Another type is the *external program engine*. This type of engine is used to support timing that is driven by events from an external program (for example, from a simulation or algorithm execution that is being visualized with an animation). Finally, a special series of user controlled timing engines are used to support interactive components. These engines are controlled and activated by user input actions.

3. PROPERTIES OF MEDIA MULTIMEDIA OBJECTS

Each multimedia object is typed and this type may imply limitations on the four components that make up the object. For example, each type typically only supports the use of a very small number of different presentation drivers (often just one). Similarly, particular types may place restrictions on the timing engines that may be used to control them (e.g. a video object may require a periodic timer with a particular frequency and an interactive object may require a user controlled timing engine of a particular type).

Although we will not explore this in detail here, adaptability to varying hardware platforms can be supported by providing multiple alternate presentation drivers (e.g. low frame rate software-only video drivers for low-end machines versus full frame rate hardware supported drivers for others). Similarly, a series of very interesting effects can be achieved by substituting alternate timing engines.

The type of an object and the four components that make it up imply certain properties and parameters that the object may have. These properties and parameters define the object's behavior and characteristics. A small set of these properties are common among all objects. These common properties are the key to the analysis and synthesis capabilities which are the central motivation for this framework.

Common properties supported by the system currently include: duration, stretchability, and flow control capabilities. For primitive objects, these properties are primarily controlled by the object's type and components. For composite objects, these properties are generally computed from the properties of the objects that they compose. Below we describe each common property.

Duration

The duration of an object defines the length of time that an object will be presented. Duration is a property of its content, but may be restricted by the object's type. The duration of an object may be in one of three states: *determined*, *not-determined*, or *indeterminate*. A duration is determined if it has been fixed to a particular value at authoring time. It is not-determined if no specific duration has been assigned. Finally, an object's duration is indeterminate if it cannot be determined at authoring time and can only be determined at run-time.

An object's possible durations are further described by three values: *natural*, *minimum*, and *maximum* durations. This treatment is similar to the boxes and glue abstraction used in the spatial domain by TeX [Knut84] and the InterViews user interface toolkit [Lint87] as well as the temporal layout algorithm described in [Buch92b]. The natural duration describes how long the presentation length would naturally last, while the minimum and maximum durations describe limits on how the object's duration may be manipulated.

Finally, each object with determined duration also maintains a record of its *scheduled* duration. This indicates the duration actually assigned to the object at authoring time.

By supporting a special infinite value for duration, this framework can model a wide range of media characteristics. A discrete media object such as a static picture would be modeled initially as "not-determined" with a minimum of zero, maximum of infinite, and natural of infinite. If the object is later constrained by a scheduling composition it might be changed to "determined" and assigned a specific scheduled duration.

An interactive component which blocks waiting for input would be modeled as "indeterminate" with minimum of zero, maximum of infinite, and natural of infinite. However, an interactive component with a timeout (such as a selector that made a default choice after a certain period) would be modeled as "indeterminate" with a fixed maximum and natural duration that corresponded to its timeout.

Finally, objects such as video clips which have an inherent fixed duration would be modeled as "determined" with the same minimum, maximum, natural, and scheduled duration values.

Stretchability

Stretchability defines the ability of an object to change its duration and the specific methods that are available to do this. Stretchability is derived from properties of both the timing engine and the presentation driver of the object.

An object can be stretchable or non-stretchable, and compressible or non-compressible. Depending on the timing engine and presenation driver employed by the object, a number of different methods could be used to stretch it. Common methods include: repeating the material, filling with blank or default material, and slowing down the presentation speed. Common methods to compress an object include: truncating the material and accelerating the presentation speed.

Each primitive object type that supports stretching (or compressing) has a default stretch (or compression) method. This default method can be changed by the author after experimentation during the authoring process.

The stretchability property is important in analysis techniques designed to detect an infeasible synchronization in a design (e.g. a non-stretchable object is required to have a duration different from its natural duration).

Flow Control Capabilities

The final common property of an object is the set of flow control capabilities that it supports. This aspect of an object is also a property of both the timing engine and the presentation driver, and indicates the ways in which the global flow of the presentation can be manipulated. These capabilities include the ability to stop, resume, abort, change direction, relocate, and change the playing speed of a presentation.

For a primitive object, flow controllability is determined by the timing engine and presentation driver in use. For composite objects, the flow controllability of the object is computed from the nature of the composition and the capabilities of the child objects. For primitive objects, each supported flow control operation is implemented by the timing engine or presentation driver internally. For composite objects the flow control operations are implemented by the composition object in terms of the flow control operations of the objects it composes (as described in Section 5 below).

4. COMPOSITION OPERATORS

In this section, we describe eight composition operators used to create composite objects. These are operators for sequential, parallel, timeline, escape, continuous synchronization, conditional, selection and repetition composition. Each of these composition operators has one or more parameters that define its characteristics.

Composition operators are hierarchical in nature. As a result each composite object is structured as a tree with the leaf nodes occupied by primitive media objects and the internal nodes by composition operator objects. The tree structure is decorated by the properties of the primitive media and the parameters of the composition operators respectively. This decorated tree structure forms the basis of the

synthesis and analysis performed within the framework.

The remainder of this section, will consider each of the composition operators of the framework in turn.

General Sequential Composition

The general sequential composition operator is used to schedule the start of one object after the start of another. This composition is controlled by one parameter — the *overlap factor*. This parameter adjusts the overlap between sequential objects. As an example, Figure 2 shows two sequential compositions with zero and non-zero overlap.

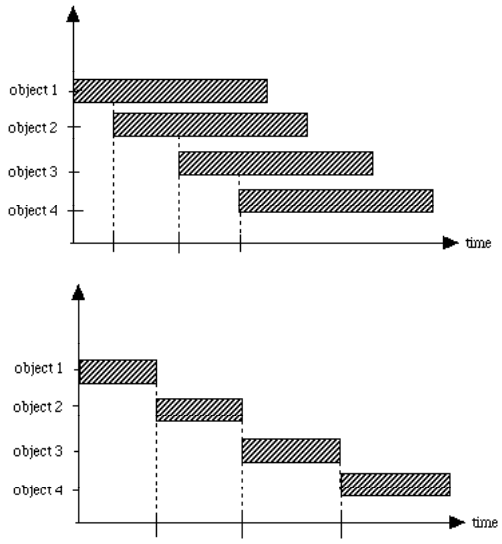


Fig. 2 Two sequential compositions.

Parallel Composition

The composite object created using the parallel composition operator schedules its child objects concurrently. The timing relationships that can be expressed among the child objects are weak in that they simply play at the same time and are not affected by what happens to the other objects. A sequence of slides shown with background music is an example application of this operator. If a certain slide is held by the user, the background music will continue to play.

One parameter controlling the parallel composition operator is the *termination criteria*. This parameter determines when a parallel object ends. It can be set to either *arbitrary* or *selected* and is accompanied by a count. In the case of an arbitrary termination, the parallel object ends when the specified number of child objects end regardless of which objects they are. In case of a selected termination, the parallel object ends only when a specifically designated child object ends. This setup covers a range of interesting termination conditions. These include a user controlled ending (e.g. via a selected termination controlled by an interactive child object) as well as termination after the first finishing object (using an arbitrary termination criteria with count 1) or termination after all objects have completed (using an arbitrary termination criteria with a count equal to the number of children).

Generalized Timeline Composition

Often one media object is used as the major presentation with other media being driven by its timing. The generalized timeline operator provides this kind of composition. Among the children of a timeline object, one is designated as the major presentation with the others subordinate to it. The timeline object schedules its major child to play at the beginning and the others according to the progress of the major child. A conventional timeline, which does not distinguish a major object from other children, can be simulated using an empty presentation with specific duration as the major object.

The relationship between the major child and the other children of an timeline object is stronger than that among the children of a parallel object. When the flow of the major child is controlled, such as stopped or aborted, the subordinate children will be affected. Figure 3 illustrates use of an animation object, an1, composed with two audio objects, au1 and au2. If the animation an1 is stopped at time t1, then the audio object au1 will be stopped and the schedule to start au2 will be delayed accordingly.

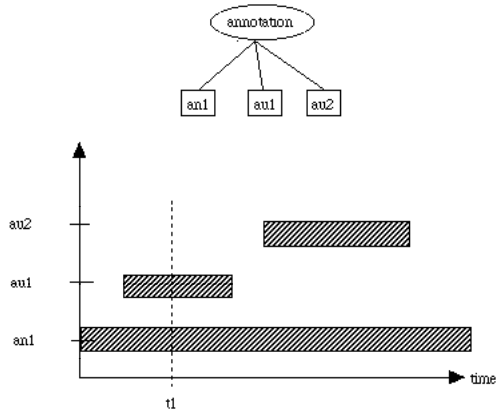


Fig. 3 An example of timeline composition.

Escape Composition

The escape composition is similar to the generalized timeline composition in that one of its children is designated as the major object. Here the major object is termed the caller and the others callees. The escape composition is different from the timeline composition in that when a callee is played, the caller is paused until the callee ends. Optionally the caller can be set to abort rather than pause. By employing interactively controlled conditional compositions (described below) this can provide the kind of "jump" semantics found in typical hypertext.

Continuous Synchronization Composition

Although the timeline and escape composition provide synchronization at start points and end points among different objects this is not sufficient in some situations. The continuous synchronization composition provides an ability to synchronize different media at a fine grain.

The continuous synchronization composition achieves this finer grain by combining the timing engines of its child objects into a single engine to drive all their presentations. Figure 4 illustrates this process. Initially, object O1 and object O2 are driven by engine a and engine b with, for example, frequency fa and fb respectively. After they are composed using the continuous synchronization composition, a new engine ab with frequency fab is created to drive both objects.

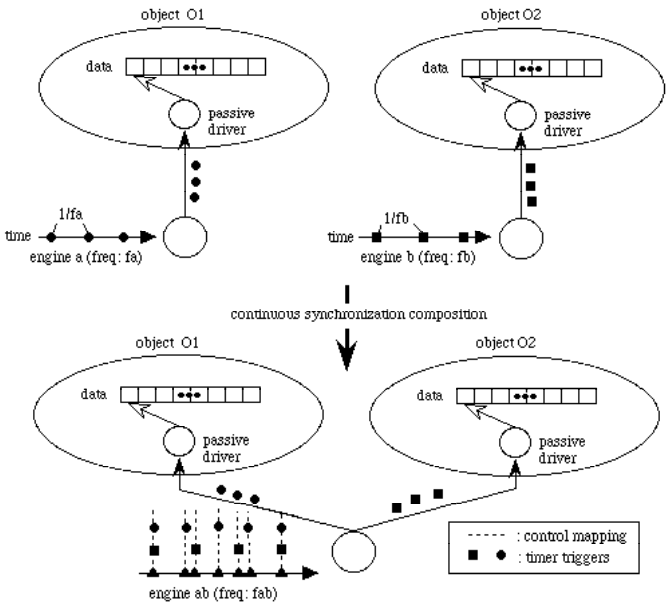


Fig. 4 Combining timing engines using the continuous synchronization composition.

Conditional Composition

The conditional composition operator provides the ability to create conditional presentations. User interaction, and the history of the presentation of other objects can be used to determine whether the composed object is presented or not.

How the operator decides whether to present its child is controlled by its *condition* parameter. This parameter can be set to operate based on several styles of user interaction or based on presentation histories. Presentation history conditions supported include comparisons against actual presentation times, the exit status of various presentations, and previously applied flow controls. Currently additional aspects of presentation status are being explored for use in conditions.

Selection Composition

The selection composition operator allows the user to choose between a number of possible presentations using several styles of user interaction. Several parameters control the behavior of a selection composition. These include the *optional* parameter which, when set "on", allows selections of "none", the *exclusive* parameter which determines whether only one object at a time may be selected, and the *reselectable* parameter which controls whether objects may be selected more than once over the lifetime of the presentation.

The user input required by both user-triggered conditional objects and selection objects is supplied by an interaction object (such as a simulated button) that is controlled and displayed by the composition operator. Two additional parameters that control the user interaction are the *timeout* and the *default action* parameters. The timeout parameter specifies the maximal waiting time of the interaction and the default action parameter defines what user action is simulated when the interaction has timed out. A blocking interaction can be achieved using an infinite timeout.

Repetition Composition

The repetition composition provides a looping ability. The parameter that controls the behavior of the repetition composition is the *repetition factor*. It specifies the number of the repetition of the composite object. This parameter can be set to infinite to support continuous looping.

5. PROPERTIES OF COMPOSITION OPERATORS

There are two properties common to each of the composition operators. These are their *schedule* and *control translation.*

Schedule

The schedule defines the actual duration and the start time for each child of a composite object. The scheduled duration for a child can be missing meaning that the natural duration is the actual duration. Otherwise, the scheduled duration will be assigned directly to the corresponding child object. Stretching or compression needs to be performed if the assigned duration is different from the natural duration.

The start times in the schedule define the time to start playing each child. They can be a constant, or an indicator of the end event of another child. In case of a constant start time, the child object will be played at that time relative to the start time of the composite object. In case of the end event indicator, the child will be started when object being waited on ends. As a result, the start times of the child objects are defined relative to the start time of the composite object, either explicitly or implicitly.

Scheduling at this level only handles the starting of objects. After that, the timing of each object is handled by its timing engine. When the engine drives an object to its end, it will remove the associated presentation and inform the top level scheduler. Under this arrangement, the top level scheduler only has to keep track of its local elapsed time and process the end events of its children in order to be able to trigger its children to start on schedule.

Control Translation

Like primitive objects, composite objects also have the ability to control the flow of the presentation. Unlike primitive objects, however, flow control commands directed a composite object need to be forwarded to the child objects. For example, stopping a composite object built from an audio object scheduled in parallel with an animation object might be done by stopping both the audio object and the animation object. However, a control command to a composite object is not always implemented by simply forwarding the same control to its children. Using the same example, if the audio object is just background music for the animation object, then the stop on the composite object can be implemented by forwarding the stop control only to the animation object without holding the audio object.

Control translation is used to enhance the flexibility of the flow control implementation of a composite object. For each type of flow control supported by a composite object, the control translation property defines the control action(s) that should be invoked for each child object This framework allows interesting and flexible flow control. For example, a composite object with an audio object, which is able jump forward and backward but can only play at a constant speed, with a video object, which can play at different speeds, can now have a fast forward control with the video playing in the faster speed, and the audio playing in a "play m out of n*m then jump forward (n-1)*m distance" fashion (where n is the speedup ratio and m is the length of an acceptable audio segment).

Both schedule and control translation are implicitly or explicitly assembled by the author during the composition process to express a design. Problems occur when a specified design cannot be honored. Possible causes include resource conflicts and

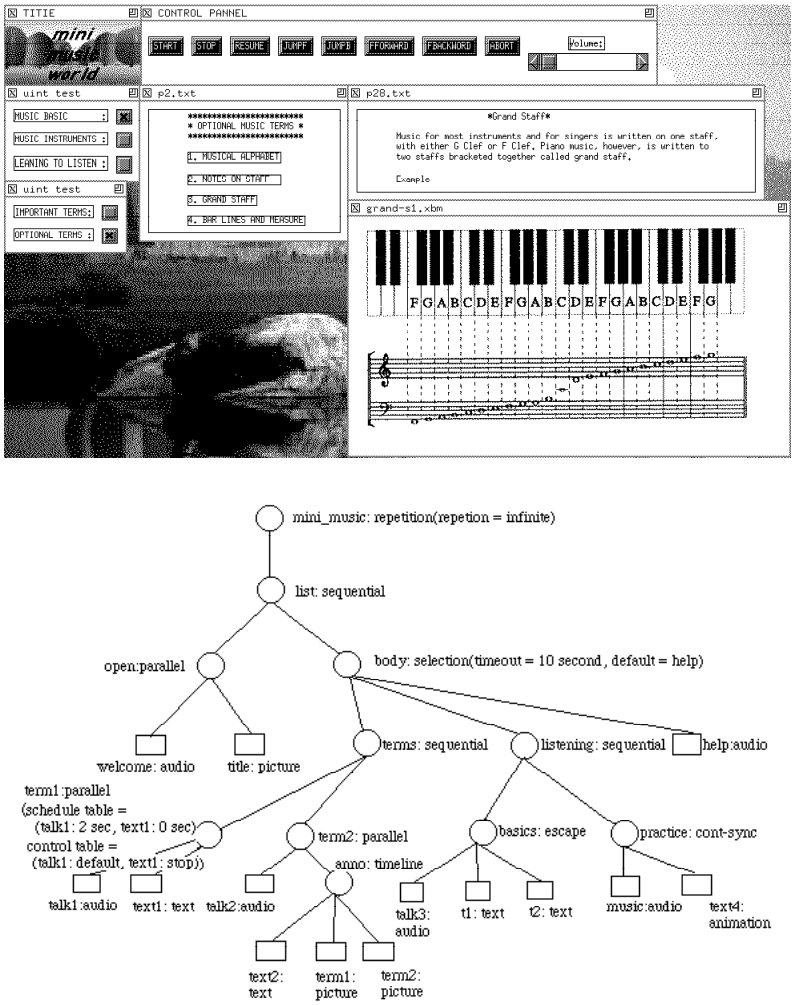


Fig.5 A multimedia document in action and its simplified structure.

attempts to stretch non-stretchable objects. Tools to locate the problem, to provide modification suggestions, or even to make the correction automatically can be made available on demand. The analysis capability of the framework which make it possible to construct such utilities for high level authoring will be discussed in Section 7.

6. AN EXAMPLE DOCUMENT

To test our framework, a small prototype system has been built for constructing multimedia documents along with a sample document which excersizes its features. This document is a small music tutor entitled "The Mini Music World". This document introduces a number of basic music terms and instruments, and teaches the correct way of listening to music.

The document in action and its simplified structure is shown in Figure 5. It starts with a welcome message and picture followed by a selection from either the introduction of the basic terms, music instruments, or instruction on listening. Each music term, e.g. term1, is introduced by a recorded audio segment with corresponding text shown on the screen. Some terms, e.g. term2, are introduced with picture annotations. Listening is taught by a recorded speech segment, talk3, with the text, t1, shown at the same time. During the speech, several terms, t2 and t3, in the text can be chosen to explore their meanings. The lesson is followed by a practice session which is composed of music and a text animation. The continuous synchronization composition is used to ensure that when a certain musical instrument starts playing, the corresponding text will be highlighted.

Properties of some basic objects and parameters of some composite objects that make this multimedia more interesting are also shown in the structure. The translation of the stop control of term1 is set to (talk1: default, text1: stop). This creates the effect that when term1 is stopped, the audio will be stopped and a default musical clip will start playing. The schedule of term1 is set to (talk1: 2 seconds, text1: 0 seconds) to start the text a little ahead of the audio for a smoother beginning. The repetition factor of mini_music is set to infinite so that the lesson will repeat forever. The timeout factor of the body is set to 10 seconds with the audio help as the default so the help message will be played when the user seems to have trouble making a decision.

7. ANALYSIS

As described in the preceding sections, a multimedia document constructed in the framework described here is expressed in the form of the tree structure decorated with properties and parameters. Although the hierarchical nature of the system eliminates some potential mistakes, errors can still occur. A dangling partial composition not attached to the main tree, objects with infinite duration but without an accessible abort control, and resource conflicts in a schedule are some examples. Fortunately, the structure developed here has been carefully designed to afford opportunities for a number of different analyses intended to help the author detect, localize, and fix the problems. In addition, some of the analysis results can be made available to the end-user to enhance the overall utility of a multimedia document before it is actually played. Information of this type which is useful to the end-user includes the types of media used in a document and the corresponding size of each type, the minimal and maximal durations to browse the whole document, and whether interaction is necessary to browse a document.

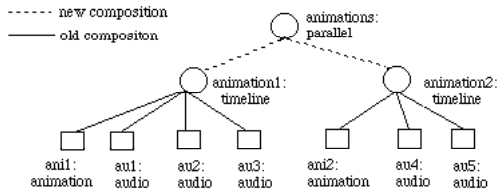
The analysis supported by our framework is based on property synthesis within media objects. During composition, the properties of a composite object will be synthesized from the properties of its children in a bottom-up fashion (e.g. resource profiles) or assembled, either explicitly or implicitly, by the author (e.g. control translations and schedules), and then if necessary, the properties of the composite object can be used to modify the properties of its child objects in a top-down fashion (e.g. assignment of scheduled durations).

In the bottom-up stage, if the synthesized properties indicate a problem, such as a resource conflict, a top-down analysis can be started to localize the source of the problem in order to provide detailed feedback to the author. Similarly in the top-down stage, if there are problems in modifying child properties, an analysis can also be started to localize the problem.

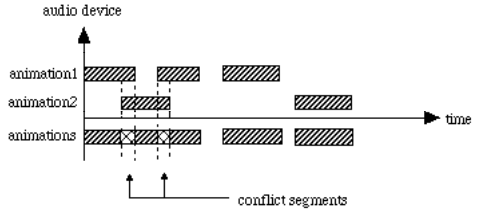
Figures 6 and 7 provide detailed examples of how error detection and localization is performed. In Figure 6(a), a parallel composition is used to compose two previously separate animations, both with a number of audio annotations. In the bottom-up analysis stage, the audio device profile in the resource property of the composition object is synthesized and the result is shown in Figure 6(b). Assuming only one audio device is available and can only be used exclusively, a resource conflict occurs. A localization analysis is therefore started to determine the specific source of the problem as shown in Figure 6(c). This information can then be used to give detailed feedback to the author about the problem and potentially to form the basis for automatically suggested solutions.

In Figure 7, a timeline composition is used to compose a video object with another parallel object. Assuming that the timeline composition imposes a longer scheduled duration for the parallel object, topic1, the duration of the parallel object needs to be changed in the top-down stage. However, a problem occurs because the parallel object does not have the necessary stretchability. In this case a localization analysis will be started from the parallel object downward and the problem will be localized to the audio object. Again, the system can use this information to provide detailed feedback and in this

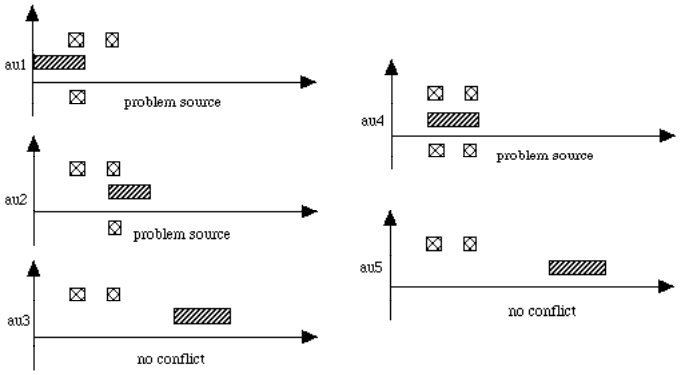
case suggest several standard ways to convert the non-stretchable audio object into a stretchable object.



(a) A parallel composition of two animations.



(b) Synthesis of the audio device usage profile.



(c) The conflict information is flowing down the tree to locate the problem sources.

Fig. 6 Analysis Example 1

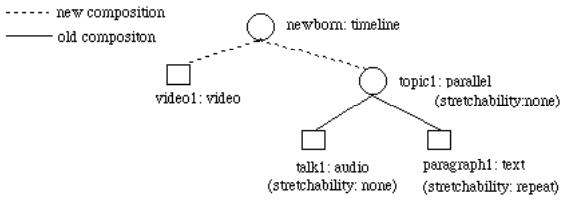


Fig.7 Analysis Example 2

As implied by the above discussion, when a problem is found during synthesis of the properties of a composite object, an analysis will be started to localize the cause(s) of the problem and inform the author. For many types of problems there will be a standardized solution schema that can be applied to fix a problem. In these cases, the system can list a set of suggested modifications or even simply choose one, allowing the author to accept or reject it. As an example, when an object with infinite duration but without an abort capability that is accessible to the user is found, an abort button can be added in the interface to fix the problem.

The analysis and synthesis capability of our framework can be integrated into a graphical editor making up part of a high-level authoring environment. For each action which causes a new composition, synthesis and analysis of properties will be started to verify the integrity of the design. If problems occur, the sources of the problems will be localized and if feasible, a number of correction can be suggested. The authoring task therefore becomes an interactive process. The author creates the multimedia documents by applying various composition operators on a set of selected media material, and the system automatically performs integrity checks and provides information and suggestions when problems occur. Problems can be found and fixed in place during the authoring process instead of after the whole composition is complete.

In addition to providing debugging aids, the analysis and synthesis capacity can also be applied to support other kinds of design aids. As we have seen in Section 4, a number of parameters need to be set when a composition operator is applied during the composition process. Although most of the parameters have default values, it is likely that the author will change them to create more interesting designs. Unfortunately, it can be time consuming to find the right value for each parameter because the value may need to be evaluated by actually playing the composition. In addition, the proposed values may cause problems, such as resource conflicts, triggering a debugging session.

Although debugging activity is necessary to experiment with different designs, information indicating potential problems in advance would be useful to keep the author from making unnecessary errors and to guide the authoring process. The analysis and synthesis capacity can help the author to choose a composition operator as well as to set its parameters in the following way. When a set of objects are chosen for composition, an analysis can be started against all types of composition operators to find out the range of the values each parameter can be assigned without causing problems. This information can be used in the authoring interface to guide the composition process.

Using Figure 6 again as an example, when animation1 and animation2 are selected for composition, the range of the scheduled durations for both animation objects resulting from the analysis on a parallel composition operator are empty sets. This implies that when the parallel composition is used, changes of the properties of the animation objects, such as shifting the start times of the audio annotations, would be necessary. This fact can be reflected in the interface to convey this situation in advance.

Using the composition of a number of audio objects as another example, from the analysis against the sequential composition, the only possible value for the overlap factor parameter is 0. The system can

therefore suggest 0 as the value of the overlap factor when the sequential composition operator is applied.

Testing and experimenting with different parts of a composition can also be done by isolating the corresponding piece (a subtree) of the composition for presentation and analysis. By intelligent use of analysis results, a set of truly high level authoring tools can be delivered. All of these advantages are made possible by the analysis capability of the framework.

8. CONCLUSION AND FUTURE WORK

In this paper, we have describes a hierarchical framework for low level analysis and synthesis to support high level authoring of interactive multimedia documents. Interactivity is achieved by treating user interaction as a primitive type of media. This allows user interaction to be synchronized and regulated with other media objects. By inclusion of the treatment of flow control and scheduling during the composition process, a multimedia document can support more flexible synchronization and control. Finally, the analysis capabilities afforded by the framework open the door to a series of high level authoring support mechanisms.

To evaluate our framework, a prototype system has been built on a Sun Sparc workstation using the Artkit user interface toolkit [Henr90]. A multimedia document applying a variety of composition operators has been created using this prototype. The results are promising. However, there is still room for improvement. We conclude this paper with issues for future work.

In our overall plan, the framework described here will provide the underlying support layer for a high level authoring environment. Although it is possible to author a multimedia document at the level of the framework we have provided, without additional aids, it will be rather cumbersome and difficult to move between the multimedia specifications and the actual multimedia presentation to make adjustments and understand the consequences of changes. A good graph editor for the specification may be better than a textual editor for illustrating the composition structure. However, switching between the specification, the presentation, and the various designing aid will still be required.

A good interactive paradigm to embed the functionality of our framework in will be critical to a successful authoring system. A two-view approach [Avra89] previously employed in user interfaces development tools is an attractive alternative, but difficult to implement in this environment because of the dynamic nature of the document being edited. We are currently examining several other paradigms for our high-level authoring environment including script-based editors [Fium87], iconic visual programming languages [Koeg92], programming by rehearsal [Finz84], and the walk-through metaphor. A number of techniques to enhance the framework described here are also in our future plans. These include: inclusion of more types of interactive objects and more flexible use of them in compositions, a macro utility for common constructions, optional automatic modification of a problematic composition, and spatial layout support.

REFERENCES

[Alle83] Allen, J.F., "Maintaining Knowledge about Temporal Intervals", Communications of ACM, vol. 26, no. 11, Nov. 1983. pp. 832-843.

[Avra89] Avrahami, G, Brooks, K.P., and Brown, M., "A Two-View Approach to Constructing User-Interfaces", ACM Computer Graphics, v.23, no. 3, July 1989, pp. 137-146.

[Buch92a] Buchanan M.C. and Zellweger, P., "Specifying Temporal Behavior in Hypermedia Documents", Proc. of the European Conference on Hypertext 1992, Milan, Italy, December 1992.

[Buch92b] Buchanan M.C. and Zellweger P., "Scheduling Multimedia Documents Using Temporal Constraints", Proc. of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, CA., November 1992.

[Camp74] Campbell, R.H. and Habermann, A.N., "The Specification of Process Synchronization by Path Expression", Lecture Notes in Computer Science no. 16, Operating Systems ed. G. Goos and J. Hartmanis, Springer-Verlag, 1974, pp. 89-102.

[Finz84] Finzer, W., "Programming by Rehearsal", Byte Magazine, June 1984, pp. 187-210.

[Fium87] Fiume, E., Tsichritzis, D. and Dami, L, "A Temporal Scripting Language for Object-Oriented Animation", Proceedings of Eurographics 1987, pp. 129-141.

[Geor91] George D. Drapeau and Howard Greenfield. "MAEstro — A Distributed Multimedia Authoring Environment", Proc. Summer Usenix Conference, 1991.

[Gibb91] Gibbs, S., Dami, L. and Tsichritzis, D., "An Object-Oriented Framework for Multimedia Composition and Synchronization", Proceedings of the First Workshop, Stockholm, Sweden, April 1991, pp. 101-111.

[Henr90] Henry, T.R., Hudson, S.E., Newell, G.L., "Integrating Gesture and Snapping into a User Interface Toolkit", Proceedings of the ACM Symposium on User Interface Software and Technology, October 1990, pp. 112-121.

[Hoep91] Hoepner, P. "Synchronizing the Presentation of Multimedia Objects — ODA Extensions", Multimedia: System, Interaction, and Applications, 1st

Eurographics Workshop, Stockholm, Sweden, April 1991, 87-100.

[Koeg92] Koegel, J.F., Rutledge, J.L and Heines, J. "Visual Programming Abstractions for Interactive Multimedia Presentation Authoring", IEEE Workshop on Visual Languages, October 1989, Rome, pp. 231-233.

[Knut84] Knuth, D. "The TeXbook", Addison-Wesley, Reading, Mass., 1984.

[Kumm91] Kummer, M. and Kuhn W. "ASE — Audio and Synchronization Extension of Compound Documents. Multimedia", System, Interaction, and Applications. 1st Eurographics Workshop, Stockholm, Sweden, April 1991, pp. 112-125.

[Lint87] Linton, M.A., and Calder, P.R., “The design and implementation of InterViews”, *Proceedings of USENIX Association C++ Workshop*, pp. 256-267, 1987.

[Pete81] Peterson, J.L. "Petri Net Theory and the Modeling of Systems",. Prentice-Hall, Englewood Cliffs, N.J. 1981.

[Stot89] Stotts, P.D. and Furuta, R. "Petri-Net-Based HyperText: Document Structure with Browsing Semantics", ACM Transactions on Information Systems, v.7, no. 1, January 1989, pp. 3-29

[Zell92] Zellweger, P.T. "Toward a Model for Active Multimedia Documents", Multimedia Interface Design, ed. M. Blattner and R. Dannenberg, ACM Press, 1992, pp. 39-52.

[Zell88] Zellweger, P., Terry, D., and Swinehart, D. "An overview of the Etherphone system and its applications", Proc. 2nd IEEE Computer Workstations Conference, Santa Clara, CA, March 1988, pp. 160-168.

[Zell89] Zellweger, P.T. "Scripted Documents: A Hypermedia Path Mechanism", Hypertext Proceedings, 1989, pp. 1-14.