Design and Implementation of an Object-oriented Media Composition Framework

Philipp Ackermann
University of Zurich, Department of Computer Science, Multimedia Laboratory
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland
e-mail: ackermann@ifi.uzh.ch

Abstract
The goal of the presented project is a general compositional environment for multimedia with interactive editing facilities. Multimedia presentations are regarded as hierarchically composed time objects that define serial or parallel synchronisation of the inserted media objects. Such media composition hierarchies support automatic temporal layout mechanisms and are integrated into an object-oriented application framework that provides direct manipulation and temporal structure through user interface components. In this paper, reusable and extendable components for audio and music processing are presented.

Keywords: multimedia composition, audio, 3D graphics, music, temporal synchronisation, animation, user interface interaction, object-oriented application framework, C++, MET++. 

1. Introduction
Today computer workstations are used to integrate different media such as text, graphics, audio, and video into one single system in which these media types are handled and manipulated fully digital. In such multimedia systems, editing of media data can be done with high accuracy, no loss of quality, and easy control possibilities. These editing features enable users to add more expressiveness into the creative process of media composition. Incorporating visual and audible media comes to interest in many fields, e.g. mixed-media art, information presentation, learning environments, audio-video post-production, data modification/animation. In order to support the creation of multimedia data, authoring systems must permit direct manipulative interaction on graphical representation of the large amount of parameters defining content, composition, synchronisation, and execution of media data. The hardware technology that is needed for multimedia applications is available today as separate components. However, programming software for multimedia applications is an ongoing task because the following requirements must be fulfilled:

- integration of several media types with different realtime constraints at low-level device interfaces
- support for media compositions to define presentations with high-level intermedia synchronisation
- intuitive user interface interactions with direct manipulation of graphical representations and high semantic feedback
- flexibility and prototypability of view of different hardware configurations and platforms, having in mind the rapid hard ware evolution and high hardware dependencies because of analog-to-digital & digital-to-analog converters, digital signal processors, and other special purpose processors.

The complexity of interactive multimedia software systems compels attention to the software engineering aspect. In this project the powerful concepts of object-oriented application frameworks are used to integrate audio-visual multimedia data types such as audio and 3D graphics into a reusable software system. An application framework is a set of interconnected objects that provides the basic functionality of a working application, but which can be easily specialized (through subclassing and inheritance) into individual applications. An application framework not only allows the reuse of code as a class library, but also the reuse of design structures, because the dependencies between objects are pre-implemented through predefined object composition, event dispatching mechanisms, and message flow control. This means that the glue between components such as media representations, real-time controllers, user interface components, media views, user actors, etc. is inherited as functionality from the framework.

2. System Architecture
The media framework is based on the object-oriented application framework ET++ [Weinand, 1989], which was developed at the University of Zurich in the programming language C++. ET++ is an object-oriented class library integrating user interface building blocks, basic data structures, support for object input/output, printing, and high level application framework components. It eases the development of interactive text/graphics applications with direct manipulation and high semantic feedback. In our project the 2D graphics model of ET++ was extended to 3D graphics, audio classes were added, and time structures were designed (Ackermann, 1993a/b). Hardware dependencies are hidden in a portability layer, that provides abstract interfaces to operating system, window system, as well as audio and MIDI UO (Figure 1). The main development platform of the MET++ (Media-ET++) project is currently a Silicon Graphics Indigo workstation.

3. Time Synchronisation
For multimedia applications time synchronisation is an important aspect. Looking at the structure in Figure 2, the music piece consists of many parts on several layers. This hierarchy contains implicit information about grouping and temporal constraints. The root node "music piece" defines part-of relations for intro, theme, and solo, meaning that they
are played in sequence. The duration of the music piece is specified by the sum of the duration of its chord objects. The chord node "Cm7" specifies a parallel temporal structure of its child objects, e.g., the notes should be played in parallel and the duration of the chord is the maximum of its child duration. Abstracting from this music example, we introduce temporal specification by hierarchical composition as a generic mechanism that includes information about part-of relations, grouping, and temporal constraints, and that allows automatic time layout calculation based on its part hierarchy. Furthermore, the composition structure can visually presented so that its implicit and often hidden temporal information will be explicit and available for direct manipulation.

```
Figure 1. The System Architecture of the MET++ Media Application Framework.
```

![Diagram showing system architecture](image)

```
Figure 2. Temporal Composition.
```

![Diagram showing temporal composition](image)

3.1 Time Events

In the media framework time layout objects and media objects are the basic building blocks of multimedia presentations, which are interrelated together in a hierarchical composition structure. The components of this structure are modeled as time events and have their own starting point, duration, and virtual time line. The time events are combined through a flexible composition mechanism with automatic object positioning along the time axis. In the time line approach, objects are attached independently from each other to a time axis, and removing an object does not affect the time position of other objects. In contrary, receiving a time event from a composition hierarchy will initiate a new position calculation so that the duration change will be propagated along the tree. Time dependent objects are realized as an object-oriented inheritance hierarchy whose base class is called TEvent (time event). All time dependent classes are subclasses of TEvent (Table 1) and inherit its synchronization features. The high-level synchronisation accuracy is based on millisecond time resolution.

3.2 Time-Dynamic Media Objects

In a multimedia presentation, the temporal relationships between media objects are configured by composing universal time objects such as TimeSequence, TimeLine, TimeSynch, TimeMix, and TimeLoop with special media objects. Any time dependent object inherited from TEvent can be inserted into such an object composition structure. There exist predefined media classes derived from TEvent for animated visual 2D & 3D objects, camera, audio, and music components (Table 1). They are realized as time wrappers to existing objects in the ETT++ class hierarchy, it is easy to extend the time framework by realizing a new time wrapper for a specific object class.

The time dynamic behavior of a time wrapper is supported through time functions that manipulate values of the controlled object. Actions such as fading, scaling, or positioning of media objects are addressed with time functions and are handled within each class independently. A time function can manipulate the time itself, so that each TEvent can specify a local (object) time, which maps the global presentation time to a specific temporal behavior. TEvent objects are synchronized to the global presentation time at the beginning and ending, whereas in this intrival, it is possible to deform the linear time progress.

3.4 Real-time Presentation

For real-time multimedia presentation, the root object, usually a Conductor object, periodically sends Perform messages to the time dependent media objects along the composition hierarchy. The
media objects compute only the data that is necessary to perform the next interval in time. Continuous media data is computed incrementally ahead into a buffer. Each media class itself is responsible for solving low-level hard real-time constraints and hides media stream dependencies (e.g. writing to device drivers, using DMA to transfer streams to DSP chips, starting a subprocess or thread, etc.).

Interactions for fast-forwarding, rewinding, or slow-motion are realized by changing the parameters of the message *Perform*. The parameters of the *Perform* message are the start time and duration of the interval that should be performed, plus the real-time interval duration. This means that each media object can adjust its behavior according to the relation between presentation duration and real-time duration. A graphical controller of the *Conductor* provides interaction for start, stop, pause, fast-forward, rewind, speed change, looping, loop points, locator points, and time position setting.

4. Visualization of Temporal Structures

In contrast to synchronisation specified by language (scripts), the benefit of hierarchical object composition having time functions is that its structure can be presented and specified graphically and allows interactive multimedia authoring with direct manipulation techniques. The time event graph (figure 3) is an obvious visualization of the temporal composition structure and reflects the part of hierarchy.

The boxes in the *event graph* represents the hierarchical structuring where the leaves of the tree are time-dynamic media objects and the inner nodes are grouping elements with a specific temporal layout. Unfortunately, the correlation between objects and the time axis is not visible. Without a time axis it makes no sense to visualize time functions. Therefore, a new time view was created, called *time composition view*, which depends on a time axis and still visualizes the hierarchical grouping structure (figure 4). It displays the temporal structure in a linearized form along the time axis in the x direction and the hierarchical composition structure in the y direction. The spatial and temporal placement of the components is automatically determined in each time object according to its child objects. Flexible direct manipulative editing features support the media creation process.

5. Audio and Music Classes

The following classes are basic building blocks for audio and music processing. The class *Intensity* is used for representing rational values for amplitude, volume, gain level, etc. It can operate on and convert itself to byte, integer, floating-point, 32bit and MIDI values. The *Pitch* class is able to handle information about pitch key and pitch frequency. It uses the *TonalSystem* class to map symbolic keys to physical cycles per seconds (Hz). The class *Beat* represents beat and bar properties and delegates quantization and the mapping from symbolic beat measure to physical time in seconds to the *MusicalContext* class. The *PitchScale* class can represent symbolic scale (e.g. chromatic, diatonic) or physical intervals for tuning purposes (e.g. equal-tempered in cents).

Audio resources are modelled with the abstract *AudioUnit* class (Table 2) as modules of a source-filter-sink architecture. An *AudioUnit* object is either a source that produces audio samples, a sink that consumes audio samples, or a filter through which audio samples flow. The output of an *AudioUnit* can be sent to different other *AudioUnits* and thus define an audio signal flow graph. This flow and its parameter can be configured in the audio system editor shown in figure 5.
Different audio file formats (AIFF, SUN/NeXT, RIFF/WAVE) are supported. Audio files can be played back in real-time from disc or can be converted to a Sound object, that holds the samples in memory. The samples can be visualized and edited in a sample view or in an envelope view.

The interpretation context of a musical performance is modelled in the time-dependent Musical-Context class. It contains information about the tonic system, tonality, signature, measure, and tempo. The MusicPlayer interprets its notes in this context and delegates the note playing to the abstract MusicInstrument class. The concrete subclasses MIDI-Instrument, CsoundInstrument, and Synthesizer map the internal note representation to device-specific synthesis parameters. The MIDIInstrument sends MIDI messages over a serial interface to external MIDI devices, the CsoundInstrument controls a sub-process that is running Csound [Vercor, 1993] as a server, and the Synthesizer dynamically allocates Oscillator objects in order to produce the sound with its own audio components of the media framework.

A MusicPlayer can switch at run-time the MusicInstrument that it will control. Converters for MIDI Files and Csound Score files are provided. A control object for a Yamaha DMR8 digital mixer/recorder was developed in order to realize mix automation within the framework. Other MIDI devices can easily be integrated.

6. Sample Applications

Different applications are under development in order to test the functionality of the time and media extensions and to improve the multimedia framework by redesign. One of these test applications is shown in Figure 5. It is a multimedia authoring tool that handles 2D drawings, images, text, 3D graphics, audio, and musical data. The hierarchical object composition ("Lego" approach) for visual and temporal structures are displayed in separate windows. Based on the functionality of the media framework, a new audio mixing metaphor was realized. Audio channels of the mixer can be attached to 3D objects, so that they depend on their position in the room and the camera view the loudness, panorama and reverb is set accordingly. This allows very intuitive arrangements in setting to music for computer animations and film post-productions and proves the additional possibilities gained by the integration of 2D and 3D graphics with audio and music components.

7. Conclusions

A large effort has gone into the design of the MET++ multimedia application framework to provide real reusable classes. This paid off by simplifying the task of writing multimedia applications and reducing development time. Although the effort to understand the framework is comparatively high, the results of projects using it are encouraging. By end of the year 1994 it is planned to release a public domain version of the MET++ multimedia application framework from ftp://ftp.uni.unizh.ch.

8. Acknowledgements

Thanks are due to André Weinand and Erich Gamma, who are the principal designers and developers of ET++. The presented work is supported by the Swiss National Science Foundation.

References


Figure 5. A media authoring tool developed with reusable components of the MET++ Multimedia Application Framework.

ICMC Proceedings 1994 223 Composition, Composition Systems