

A SCRIPT BASED APPROACH FOR INTERACTIVE MULTIMEDIA APPLICATIONS

MICHAEL VAZIRGIANNIS, MICHAEL HATZOPOULOS
*Department of Informatics, University of Athens,
Ktiria Typa, Panepistimiopolis, 15771 Ilisia,
Athens, Greece.*

ABSTRACT

This paper proposes an object oriented framework for scripting Interactive Multimedia Applications. Unlike other existing systems, the script (scenario) concept is central for the design since it handles the composition of the media objects both in spatial and temporal domains, the user interaction and system events. Moreover the proposed model incorporates an advanced scheme for complex composition in the temporal domain using a complete set of temporal operators defined in the framework. A sample application illustrates the potential of the proposed framework.

1. Introduction

Multimedia presentations are evolving as a popular and demanding field since the potential of new multimedia technologies is challenging a variety of group of users. We aimed at defining an object oriented framework that abstracts the composition and synchronisation features of an Interactive Multimedia Application (IMA) and, moreover, provides the methods for the definition and execution of an interactive scenario. Special emphasis is put to the scenario concept, since an IMA may be conceived as a theatrical play which consists of scenes, there is a script for each scene etc. The only difference is that an IMA provides interactivity features (i.e. the user may alter and moreover define the flow of the IMA according to the scenario of course).

The paper consists of three parts. In the first part, there is a reference to related work done in our department. The previous work refers to efforts for modelling spatial and temporal composition of multimedia objects¹³ and extensions to a parallel language for supporting real-time applications¹¹. Also we review important concepts of multimedia composition and synchronisation. There are sections referring to the actions that may be applied on multimedia objects (in space and time domains), the temporal relationships among actions and the synchronisation between them. In the second and third part, we

define the object oriented framework, analyse and evaluate the design through an indicative sample application. The model is defined in a notation similar to C++ and is based on previous pieces of work done in our department. Finally in the conclusion, we attempt to evaluate the proposed model in terms of the objectives initially set as well as in comparison to other systems referred in the background information. Also we propose future research directions.

2. Background Information

In our department we have defined a data model that aims at representing the features and functionality of multimedia composition in space and time¹³. It is implemented as an object oriented hierarchy, whose base class is called `item`. Multimedia information is modelled by the media classes, subclasses of `item`. Each of the media classes inherits all the data and behaviour of the `item` class, while it defines the additional attributes and methods that are specific to each media type. The composition of multimedia in space and time was modelled by the class `composite_item`. Also in the department there is active research on language extensions to support the implementation of real-time (RT) applications¹¹. In addition, the development of complex interactive multimedia applications presents similar characteristics and requirements with the RT applications since a desirable sequence and ordering of a set of actions in the time domain is required. For this reason much of our previous work in the field of RT processing has been incorporated in the model that follows. In the following there is an analysis of the several aspects of complex multimedia presentations. These aspects are:

- actions and temporal relationships among them
- multimedia synchronisation
- multimedia composition (spatial and temporal)

2.1 Actions and Temporal Relationships among them

The manipulation of multimedia objects is merely based on the actions that modify the spatial, temporal or storage features of them. Hoepner⁴ defines actions as arbitrary acts that are classified into atomic and composed ones. Atomic actions cannot be subdivided into partial actions for the purpose of synchronisation while composed actions consists of atomic or other composed ones whose parts have to be synchronised. This classification is made mainly for synchronisation purposes so that atomic actions are used for synchronisation scenaria. The start and end points of an action are used as synchronisation points. Another classification of actions is attempted by

Little² where the actions are classified into unary (that adjust the multimedia objects according to the presentation requirements) and binary (that compose the adjusted objects according to the presentation script).

The topic of temporal relations between actions has been addressed by Allen⁶. In this paper there is a definition of a complete set of temporal relations between two actions. These are: *before*, *during*, *overlaps*, *starts*, *ends*, *equal* and the inverse ones (this does not apply to equal). Adding vacant time intervals the set is extended with the sequential, parallel first and parallel last relations. Hoepner⁴ defines a set of path operators that are based on the aforementioned relationships. This set defines the semantics of a synchronisation mechanism and the synchronisation of the presentation of multimedia objects. The most important operators which are also incorporated in the proposed model are: \wedge , \vee , $*$. They are defined as follows:

- $A \wedge B$ (parallel-last): Actions A and B are started at a common start point and are executed concurrently. The composed action terminates when all the participating actions (A and B) terminated.
- $A \vee B$ (parallel-first): Actions A and B are started at a common start point and are executed concurrently. The composed action terminates when the first in time participating action (either A or B) terminates.
- A^i (repetition) : Action A will be repeated i times.

2.2 Multimedia Synchronisation

The concept of synchronisation in the context of multimedia information systems refers to the mechanisms used by processes to coordinate the ordering in time domain. According to Gibbs¹ the objectives of multimedia synchronisation are: starting and stopping multimedia objects at desired time points, establishing or removing connections between components at transition points and ensuring global synchronisation between activated components.

The main concepts introduced by Gibbs¹ to achieve synchronisation are the *world_time* and *object_time* that are necessary in defining a low level synchronisation scheme. The *world_time* concept is defined as a reference time and refers to the time elapsed from the origin of time, which is usually the start of the current application. The *object_time* concept refers to a multimedia object internal time. In general, the synchronisation of multimedia objects presentation has two aspects:

- low level synchronisation, which addresses the problems of hardware capability for preserving certain temporal features of multimedia composition within a range.
- high level synchronisation, which refers to the sequence of actions on multimedia objects and handling of events that occur and cause other actions.

The proposed model does not deal with the low level synchronisation since, the objective is the definition of a generic, abstract design for modelling the functionality of multimedia composition and synchronisation. Moreover, it is assumed that the underlying hardware and software is capable of proper execution and presentation of the composed multimedia objects.

Another effort for modelling composite multimedia is the evolving standard MHEG (Multimedia and Hypermedia information coding Expert Group)¹². As regards temporal composition MHEG defines structures that support three types of synchronisation:

- elementary synchronisation, which refers to two objects or actions (O1, O2) and the respective time intervals (T1, T2) The elementary synchronisation is classified into sequential (O1 is executed T1 time units after the application start, O2 is executed T2 time units after the end of O1) and parallel (O1 is executed T1 time units after the application start while O2 is executed T2 time units after the application start)
- chained synchronisation, in which there is a set of objects (O1, O2,..) and the respective time intervals (T1, T2..). For instance O1 is executed T1 time units after the application start, O2 is executed T2 time units after the end of O1, O3 is executed T3 time units after the end of O2 and so on.
- cyclic synchronisation, in which a set of events is repeated at specific time intervals.

2.3 Multimedia Composition

The most important aspects in the composition of multimedia objects are the spatial, and temporal ones¹. Spatial composition refers to the spatial features of multimedia objects. The spatial relationships of objects are generally expressed by their location in space (set of coordinates in space at which a given action may take place) while the temporal relationships are expressed in terms of time coordinates (set of values in time where a set of actions may happen) for the objects⁴.

Temporal composition refers to the temporal relationships between objects that participate in a complex presentation. Temporal composition also

refers to real-time behavioural properties of the objects since there is a need to define timing constraints for the execution of an action or define the maximum time interval for an object to wait during synchronisation. A more detailed discussion about the introduction of the time in programming systems as well as the problems of the verification and validation of such systems can be found in Levi⁹ and Stankovic¹⁰.

An important concept in multimedia presentations is the scenario which may be viewed as a predefined spatial and temporal sequence of presentation of multimedia objects. It may either be static or dynamic (i.e. the flow of actions may be modified according to events that may happen and are accounted for the scenario script).

In the commercial field MACROMIND DIRECTOR is one of the leading tools for composite multimedia presentations. It runs on APPLE Macintosh platforms and is used for the preparation of interactive multimedia applications. It internally uses the LINGO language⁷ which among other features supports temporal composition of media objects. Temporal features of LINGO deal with timer manipulation (there are system timers and a general timer that using the timeOut feature generate events). LINGO also supports event manipulation. There are certain event types: keyDown, mouseDown, mouseUp that are recognised as events and invoke the so-called event scripts or macros. Also the timeout event which is caused when one of the aforementioned timers reaches the timeout point set by the system or the user. LINGO also provides time manipulation language constructs.

Another formalism that is proposed for representation of hypermedia documents is HYTIME³. It is a structuring language based on SGML which aims at representing hypermedia documents at a high level thus not concerned with the underlying formats and synchronisation details.

3. The Proposed Model

We aimed at the definition of an object oriented framework for representation and manipulation of IMAs. The main attractive point of the design is modelling of the scenario concept which describes the flow of the application in space and time and handles the user interaction and unlike other existing systems interrupts, system events, exceptions. Another innovative feature of the proposed system is that each active element of an IMA may have a script (an instance of the `scenario` class) defining its functionality. This is a desirable feature which results in a flexible and reusable approach.

An important construct of the proposed design is the action concept, its definition and classification. As actions we accept acts which aim to manipulate spatial or temporal features of one or more multimedia objects. In the proposed model three temporal operators are used in order to construct complex (n-ary) actions using elementary (unary) ones. The temporal

expressions generated in this way define complex actions which include the notion of synchronisation between the elementary ones. The three temporal operators which have been already defined are: \wedge (parallel-last), \vee (parallel-first) and $*$ (repetition) operators.

Another important concept in the proposed model is the notion of the events. Events are arbitrary distributed points in time used for the ordering of the actions and their coordinated access to shared resources. A similar approach for synchronisation of multimedia systems has been proposed in ⁵. Events may be regarded as a special kind of information passing mechanism between objects, since the information transmitted is only the start and end of a specific action. More precisely, within the lifetime of every object there are two special events which are generated:

- the start point of a specific action.
- the end of that action.

If two or more media objects are combined in a multimedia presentation the above events will be used in order to assure the desirable temporal order of presentation. For example, if an event which has been generated at the end of a specific action is used to trigger the execution of another action, then we have the "sequential" synchronisation. If the event which is generated at the start of an action is used to trigger the execution of another action, then we have a "parallel" synchronisation. More complex synchronisation conditions can be formed if more media objects are combined for a multimedia presentation.

Moreover, selecting a specific button, during a presentation, the corresponding event is caused. In this way, as it will be described in the following paragraphs, asynchronous interaction with the user can also be handled by the model. Also two special events (`timeout` and `no_sync`) can be produced in case of a time constraint violation or from synchronisation problems.

As already mentioned, the proposed design does not involve low level synchronisation issues regarding the capability of the hardware for proper presentation of data intensive multimedia information. For the proposed design, synchronisation refers to the level of sequence of actions and handling of events.

Handling of interrupts is also supported by the model. When we refer to interrupts we mean the asynchronous control signals that can be sent to the application by pressing special control keys (e.g. $\wedge C$) or by a hardware failure.

The model is realised as an object oriented hierarchy of classes. The class hierarchy contains two categories of classes. The first one is the hierarchy of the classes for multimedia data representation and manipulation. The second includes classes for the representation and manipulation of the spatial and temporal features of an IMA. The proposed design is not implemented yet,

though it serves as a generic design that has taken into account implementation issues.

3.1 Multimedia classes

The base of all the multimedia classes is the class `item`. This class serves as the root of the entire hierarchy of the proposed model and includes generic data variables and methods. Data variables which are specific to each media type as well as the corresponding methods are included in the media classes which correspond to each media type. Member variables are included for administrative purposes like the identification of the item (`oid`), a short description of the item (`name`). The hierarchy in fig. 1. refers to the classes for multimedia manipulation. A detailed description of those classes is not presented here for space reasons since they are just abstractions of the well known media functionality.

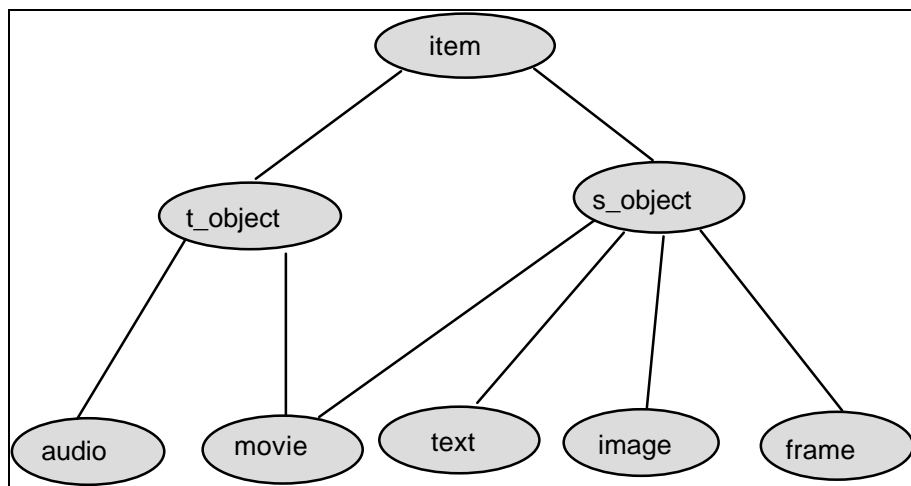


fig. 1. Media representation classes

The multimedia objects may be classified in two categories: the spatial (text, graphics, images, movie) and the time dependent ones (sound, movie). The spatial media objects are represented in the proposed model by the class `s_object`, which is an abstraction of the features and functionality of the spatial media objects (text, movie and image). Its member variables include the dimensions of the spatial object : `x`, `y`, `w`, `h` (where `x`, `y` is the origin of the `s_object` relative to the origin of the frame which includes the `s_object`, while `w`, `h` are its dimensions), the plane on which the spatial object resides (`plane`) and the transparency factor (`transparency`) for the cases in which the current object is overlaid with another. In many cases presentation of multimedia information involves scaling in space and/or time. As regards the spatial domain the member variable `s_scale_f` fulfils the aforementioned requirement. The member functions manipulate the presentation features of

an `s_object` and range among displaying (`display()`), hiding(`hide()`), moving the object backward (`backward()`) or forward(`forward()`) one or more planes in a frame etc.

As for the time dependent objects the `t_object` class is an abstraction of the features and functionality of the temporal media objects (sound and movie). The temporal features of a multimedia object are supported by the `object_time` data variable and the methods that transform and manipulate it. The `object_time` variable measures the time and starts as soon as the object is activated, pauses when the object is suspended and becomes null when the object is stopped. The `object_time` is proportional to the percentage of multimedia object's data that have been processed. The `status` data variable indicates the status of the item which may be either active, idle or suspended. It is used mainly for global synchronisation in the case of complex multimedia presentations. The `t_scale_f` variable is the temporal scale factor of the item execution. By default it is set to 1. The member variable `direction` indicates the direction of playback for the time variant media. By default it is set to `F`. If it is set to `T` then the media object is played back in the inverse direction.

The methods of the `t_object` class manipulate the temporal features of a multimedia object. The method `cue(ts,td)` is used for preparation (usually buffering) of a temporal portion of the object starting at `ts`, lasting `td` time units. The method `jump(ts)` moves the file pointer indicating the current position in the temporal space, so that the presentation point moves relatively by `ts` time units. This method is mainly used for synchronisation purposes. The methods `start()`, `stop()`, `pause()` and `resume()` are used for modelling the corresponding actions which correspond to common functionality for time variant media. Each of these methods changes the status of the item accordingly. It is essential for synchronisation purposes to be able for the application to know the status of the item and the duration of item's execution at normal speed (`t_scale_f = 1`) whenever applicable. These needs are facilitated by the `get_status()` and `get_duration()` methods.

Time transformation methods are essential for synchronisation purposes when there is the need for communication between the application and the object. The methods `t_translate(World_time)`, `t_scale(f)`, `t_Invert()` implement the corresponding temporal transformations. The first translates the objects execution in time by `World_time` time units on a virtual time dimension. The second scales the duration of the object's execution by a factor `t_scale_f`. The third inverts object's execution when there is such a prompt by the user or the application.

3.2 Composition Classes

The requirements for composition of multimedia are fulfilled by the set of classes (composition classes, fig. 2.) which is further described below. These requirements are:

- modelling of the composition of multimedia objects in space (`s_composite`, `frame`, `scene`)

The spatial composition is defined by the class `s_composite` which includes the spatial objects that are included in a scene and their spatial relationships and transformations (i.e. position in the application window, overlapping features, spatial transformations, effects etc.)

- modelling of the composition of multimedia objects in time (`t_composite` class)

The `t_composite` class includes the identification of the time dependent media objects that participate in the temporal composition. In each instance of this class there are also the temporal specifications of the media objects to be used in the composition and the temporal transformations that will be applied on them.

- modelling of the script that defines the flow of the IMA in space and time as well as handling the user interaction, system events, exceptions and interrupts (`scenario` class)

From the above it is evident that the instances of classes `t_composite`, `s_composite` are not interrelated since they include different types of information and fulfil different requirements. Moreover, it is necessary to stress the distinction between the `t_composite` and the `scenario` classes. The former essentially specifies which time dependent media objects will participate and how (temporal portions and transformations) in a scene while the latter specifies the flow of the IMA in space and time as well as handling the user interaction, system events, exceptions and interrupts.

A more detailed description of the aforementioned composition classes (fig. 2.) follows. Starting from the `t_composite` class it is clear that it aims at representation of the composition of time dependent media objects. The `t_composite` class includes as member variables the lists of the media objects that participate in a specific scene of the application (`audio_list` for the sound objects and the `video_list` for the movie objects). Each element of these lists includes the identifier of the object (`oid`) the direction of playback during the presentation (`direction`), the scaling factor in time (`t_scale_f`) and the temporal starting (`t_s`) and ending point (`t_d`) of the object's part that will be included in the scene.

As for the composition of multimedia in the spatial domain the `s_composite` class is used for the representation and manipulation of the media objects in spatial domain. Each `s_composite` object may contain several frames and buttons at the desired positions. As for the class `frame`, it serves

presentation purposes. It includes a list of `s_objects` to be presented. It is a friend of all media classes (i.e. it has access to their data members and methods). Since it is a child of class `s_object` it inherits all the variables and methods from it so a frame has all the functionality of an `s_object`. There are some additional member variables: `v_scroll` and `h_scroll` which indicate whether the frame will be scrollable or not and the list `oid_list`, that includes the list of the identifications of the `s_objects` that participate in the application.

The most important part of an interactive multimedia application is the scenario or script concept. The flow of an IMA in space and time as well as handling the user interaction, system events, exceptions and interrupts are modelled by the `scenario` class. An instance of the `scenario` class consists of a set of tuples each one describing an action that will be performed when a specific event occurs (start of end of the execution of media objects or user action or system event). The tuple also defines the duration of the action, the synchronisation events that will be generated from the execution of this tuple, and the way that the system will handle exceptions or system interrupts during the execution of this tuple. Many of the attributes of each tuple have been introduced in programming languages for the implementation of real-time applications (Ishikawa⁸, Mourlas¹¹).

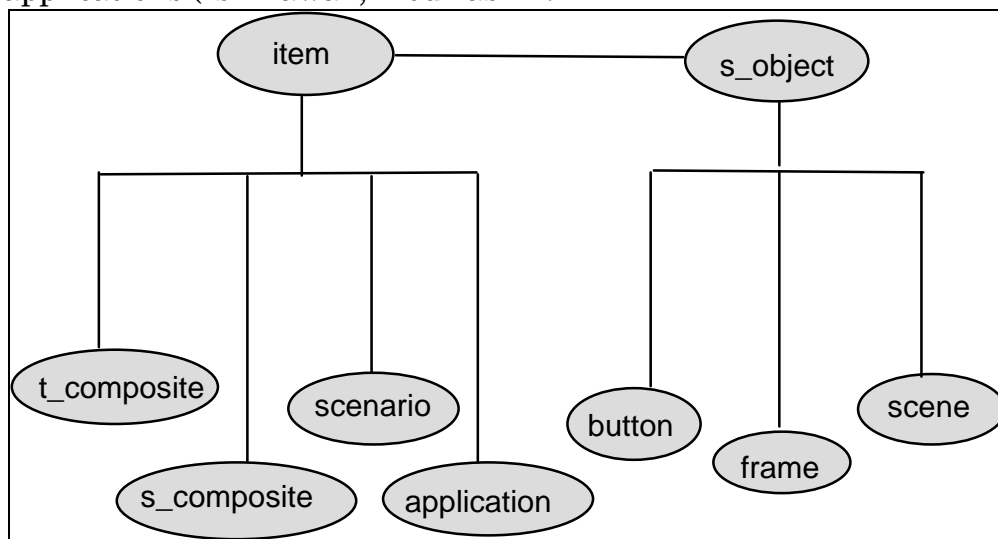


fig. 2. the classes that aim at multimedia object composition in spatial and temporal domains (composition classes).

Each tuple of the scenario list has the form :

```
<start_time, duration, <action>, synch_events, exception_handler,
interrupt_handler>
```

where:

- `start_time` determines the start of execution of the actions described in the tuple. It may be of the form: `(Abs_time and Event_tree)` or `(Abs_time or Event_tree)`. In this expression the starting time may depend either on absolute time (`Abs_time`) or on sequence of events (`Event_tree`) or on a combination of both. The sequence of events is a general logical expression of events. In this context, events are used for the ordering of actions in time domain. As already mentioned events are caused either by the start or the end of every action or by a user interaction event (e.g. selection of a button). An event may cause the execution of a unary or n-ary action. For instance, when we need a task to start at 12:45 or at the moment that either `event1` or `event2` has happened the value of `start_time` will be `(12:45 or (event1 or event2))`.
- `Duration` determines the duration of the action in the tuple. It may be expressed in absolute time units or as a set of events that will cause the execution of the tuple to end. Similarly to `start_time` it has the form: `(Abs_time and Event_tree)` or `(Abs_time or Event_tree)`.
- `<Action>` is the list of actions that will be performed and may refer either to an individual object (in this case actions are multimedia classes method calls e.g. `play`, `move`, `display`) or to a combination of objects (like `overlay`, `overlap`, `resolve occlusion`, `scaling`, `cropping` etc.). Moreover we can construct complex actions using elementary ones and the temporal operators defined earlier. These expressions of actions include the notion of synchronisation between the actions implicitly resulting to a more elegant design of the whole presentation. In addition, with these temporal expressions we reduce also the number of events used for explicit synchronisation with the result of a simple description of a complex multimedia application. For example, the composed action `(video.play /\ sound.play)` implies that the video and the sound will start at the same time and be executed concurrently. The composite action will be terminated when both the two participating actions will be terminated.
- `Synch_events` refers to the naming of the two special events which are generated at the start and the end point of the describing `<action>`. It is an ordered pair of event names where the first name represents the event that will be occurred at the beginning of the execution of the defined `<action>` and the other is the name of the event that will be occurred at the end point of the action. These names of the events defined in this field are used for the explicit synchronisation of the actions participating in a scenario. More precisely, we use these names in the expressions `start_time` and `duration` as we have already seen in the description of these fields above. For example, the pair `(e1,e2)` for the composed action `(video.play /\ audio.play)` means that when the `start_time` expression is evaluated to

true then the event `e1` will be generated. At the end of this composed action the event `e2` will be also generated. If we do not care about the generation of an event we place an underscore (`_`) instead of its name.

- `exception_handler` is a set of actions that will be executed in case of violation of the predefined time constraints or when there exists a lack of synchronisation between the participating actions. As a result, the execution of the specific action will be suspended and the `exception_handler` will take immediately priority. This action will handle the exception and will try to bring the whole system to a desirable state. One of the two events `timeout` or `no_synch` will have occurred showing that the exception produced from a timing constraint violation or from synchronisation problems of the action respectively.
- `interrupt_handler` is an action that has to be performed immediately after a system failure or by typing special control characters (e.g. `^C`). Such failures can be produced for instance from the hardware, resource failures, environmental factors etc. and they are application independent. When a failure occurs the system receives an identifier of the interrupt which is usually a small integer and then calls this interrupt handling routine. After the execution of the handler the execution resumes at the point where it was interrupted.

Like in a theatrical play the IMA consists of several scenes. We conceive scene as the basic component of an IMA. It includes `t_compo`, an instance of the class `t_composite` and `s_compo`, an instance of the class `s_composite`. It also includes a script (instance of the class `scenario`) which defines the functionality and the interactive features of the scene.

An IMA is modelled by the application class. It includes the list of scenes (`scene_list`) that comprise the application and the scenario of the application (`script`) which is an instance of the `scenario` class. It also includes administrative information such as the author (`author`) and the creation date of the application (`creation_date`) as well as a parameter indicating the status of the application (`status`).

One of the main elements that handle the user interaction are the buttons. These elements are modelled by the `button` class. It is a child of class `s_object` and the additional members refer to the string that is displayed in the button (`name`) and the list of events that the button may accept (`events_list`). If this list is empty the button may accept any event that occurs within its boundaries and has been defined by the script.

4. A Sample Application

An IMA in the proposed model is represented by an instance of the class `application`. It consists of a set of scenes and a scenario script that defines the IMA's features. Each scene consists of an instance of the class `s_composite` (specifying the spatial composition features of the IMA), an instance of the class `t_composite` (specifying the temporal composition features of the IMA) and an instance of the `scenario` class which defines the flow of the IMA in space and time as well as handling the user interaction, system events, exceptions and interrupts. Each `s_composite` specifies which spatial media objects (text, graphics, images, spatial aspects of movie) will participate and their spatial specification for the current application (position, scaling, transformations, effects etc.). As for the `t_composite` they include the identification of the time dependent media objects that participate in the temporal composition. In each instance of this class there are also the temporal portions of the media objects to be used in the composition and the temporal transformations (scaling, inversion etc.)

We assume that there is the need for an interactive multimedia presentation concerning the Greek island Crete. The required multimedia material is available in a multimedia data base under the aforementioned multimedia data model representation scheme. The presentation refers to historical, tourist and physical information on Crete and consists of several scenes composing a hierarchy. Here we will present the introductory scene `comp1` and one of the others that are invoked from this, the one referring to the physical aspects of Crete. The presentation layout of the scenes `comp1`, `PI1` may be seen in fig. 3. The description of the objects participating in this sample application may be found in Appendix A. The application described below is not implemented, it is rather a design based on the proposed framework.

The application is represented by the instance `Crete` of the class `application`. The application contains the scenes `comp1`, `PI1`, `TI2`, `HI1`, `TI3`, `TI4` and the scenario `script_crete`. The scene `comp1` consists of the `t_comp1` instance of the `t_composite` class and the `s_comp1` instance of the `s_composite` class as well form the script `script_comp1`.

The `t_comp1` object describes the composition of the participating objects in the temporal domain and includes the sound object `crete_audio`, which will be played at normal direction and speed using the first ten seconds of the sound object. The `s_comp1` object describes the composition of the participating `s_objects` in the spatial domain and includes the frame `c1` and the buttons `b1`, `b2`, `b3`, `b4` at the defined positions.

The frame `c1` includes the image `crete.img` at normal scaling, it is scrollable both horizontally and vertically. The scenario `s_comp1` defines the functionality of the scene `comp1`. It consists of six tuples each one having the structure as described in the description of the class `script`. The first tuple indicates that the `crete_audio` sound will be played starting from the origin of the application (0) or when event `e2` occurs (`e2` is the event that is generated

by the end of the `crete_audio` object playback). Namely the `crete_audio` is played back repeatedly. The music execution will last until the close button or if `^C` key are pressed. In the former case scene's execution is stopped while in the latter only the music execution stops. The second tuple indicates that the `crete_map` will be displayed using the dissolve effect at the third second of the composite's execution. The following three tuples are the invocation of the three scenes `HI1`, `PI1`, `TI1`. The last tuple indicates that if the close button is pressed the execution of the whole composite stops.

Scene `PI1` refers to the interactive presentation of information concerning physical aspects of Crete. The scene `PI1` consists of the `t_PI1` instance of the `t_composite` class and the `s_PI1` instance of the `s_composite` class as well form the script `script_PI1`. The `t_PI1` object describes the composition of the participating objects in the temporal domain. The sound list of `t_PI1` includes three tuples. The first refers to the sound object `anim.aud` which will be played in normal direction and speed (`direction = F`, `t_scale_f = 1`) using the whole duration of the object (`ts=0`, `td=_`).

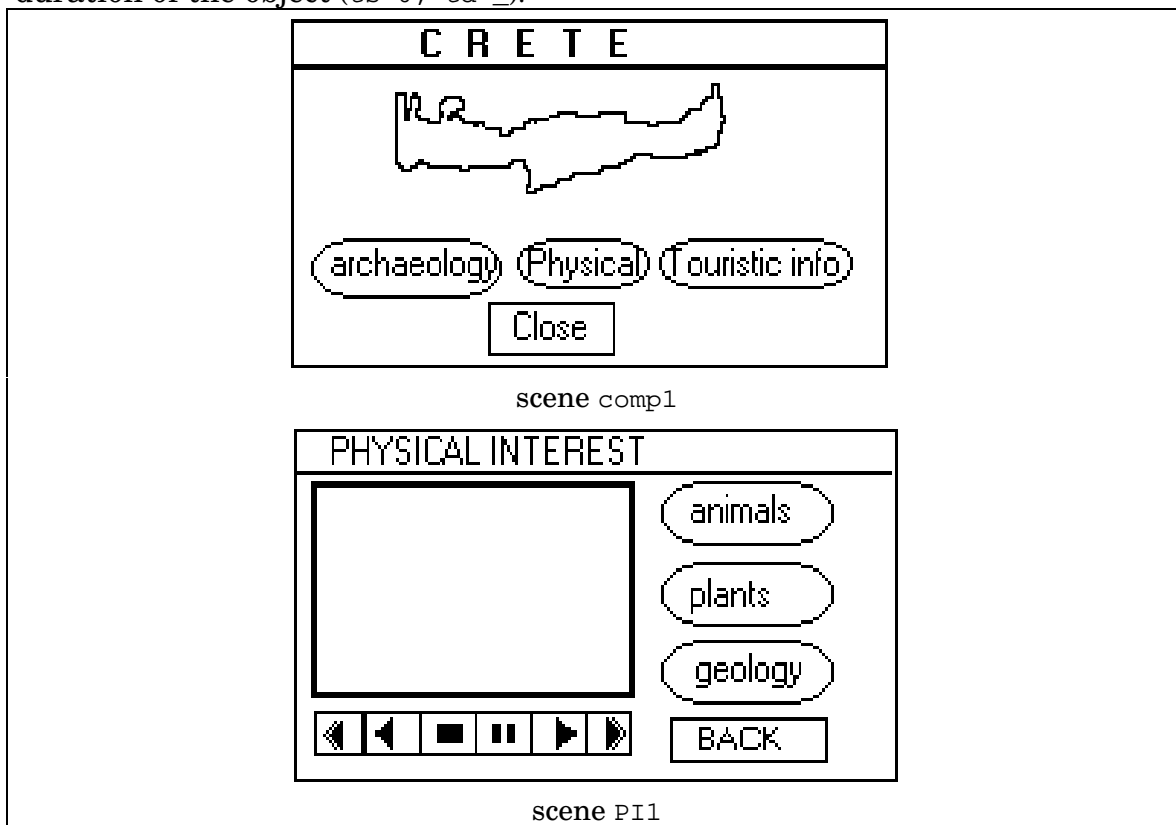


fig. 3. The presentation layout of the scenes `comp1`, `PI1`.

The same apply to the other participating sound objects (`plants.aud`, `geology.aud`). The movie list of `t_PI1` includes three tuples. The first refers to the movie object `anim.mv` which will be played in normal direction and speed (`direction = F`, `t_scale_f = 1`) using the whole duration of the object (`ts=0`,

td=_). The same apply to the other participating movie objects (plants.mv, geology.mv).

The `s_PI1` object describes the composition of the participating `s_objects` in the spatial domain and includes the frame `PI1` in the `frame_list` and the buttons for controlling the movie playback as well as the buttons `animals`, `plants` and `geology` in the `button_list`. The frame `PI1` includes the movie objects (`oid_list = anim.mv, plants.mv, geol.mv`), it is not scrollable both horizontally and vertically (`v_scroll = 0` and `h_scroll = 0`), at normal scaling (`s_scale_f = 1`). The other parameters that are not set by the definition of the `PI1` frame have the default values set in the default constructor.

The scenario `script_PI1` defines the functionality of the scene `comp1`. It consists of several tuples each one having the structure as described in the description of the class `scenario`. The first tuple indicates that the scene `PI1` will be executed until the event `back.button_up` occurs. The second tuple indicates that when the event `mv_play.button_up` occurs the movie object `anim.mv` will be started and will be played back until either on of the events `e4` (event generated by the interrupt `^C` defined in the interrupt handler of the same tuple) `mv_stop.button_up`, `vd_pause.button_up` occurs. The beginning and the endpoint of the `anim.mv` generate the events `emvst` and `emvend` respectively.

The next scenario tuple indicates that the sound object `anim.aud` will start when one of the events `emvst`, `eauend` occurs and stops when the `emvend` occurs (i.e. when the movie object `anim.mv` stops. The beginning and the endpoint of the `anim.aud` generate the events `eaudst` and `eaudend` respectively.

5. Conclusion

An effort for modelling IMAs should be abstract enough so as to represent the functionality and the data required for a generic platform independent model but also specific enough and technically complete so that this model may be instantiated on various implementations. The innovative features of the proposed model compared to existing approaches and tools (e.g. MACROMIND DIRECTOR) are:

- temporal operators between actions

The proposed model, unlike other approaches, introduces the notion of explicit high level synchronisation between time dependent multimedia objects. The expression that may be formed do not take in account the quantitative temporal features of the media objects but only the desired temporal composition features. Moreover the expressions may be complex involving may objects and sets if actions and composite actions. For instance, the expression $(A \setminus B) \wedge (C \setminus D)$ introduces a composite action in time which will be accomplished when the temporally shorter of the expressions $(A \setminus B)$, $(C \setminus D)$ is accomplished.

- scenario scripts attached to elements of the multimedia application. Definition and execution of scripts that define the flow of the application, the spatial and temporal features of the composition. These scripts are attached to all the objects that are constituents of an IMA (unlikely to MACROMIND)

- fully object oriented system

The proposed system is totally based on the object oriented approach gaining thus all the well known advantages (reusability, extensibility). Thus objects defined under the proposed model are reusable by more than one IMA.

- user defined events and uniform way of event handling

The proposed model supports user defined events apart from the traditional events (start, end) for single actions. Such events could be any specific time indication (e.g. 12:31pm) the start or end of any composite action using the temporal operators etc. Also the proposed framework supports complex logical expressions between events For instance the expression (event1 and event2) or (event3 and event4) may be used to serve as the start time or the duration of an action in a scenario tuple.

- interrupt & event handling

Handling of system errors and events is embedded in the IMA, making thus the design even more robust and platform independent.

- platform independence

Since the entire application can be represented in terms of the object oriented classes and scenario tuples of the proposed model, it is clear that the IMA is portable across platforms assuming that the appropriate interpreter and multimedia assets exist on these platforms.

As regards comparison to HYTIME, certainly the proposed approach is more limited than HYTIME. Though it differs from it in the sense that there is more emphasis put on scripting rather than embedding the the behaviour in the objects. Moreover the proposed model deals explicitly with high level synchronisation (by introducing the expressions that contain actions and temporal operators).

This design may be applied in a variety of application fields where interactive multimedia presentations are desired. Such applications may be educational, commercial advertisements, information points etc. A long term objective is the definition and implementation of a language for authoring complex multimedia presentations based on the model that has been described. Moreover a powerful editor could be built on top of the proposed model that could be used for interactive IMA editing and execution. It would

also be desirable that the developed IMA be exported in the aforementioned script language.

References

1. S. Gibbs, L. Dami and D.Tsichritzis, *An Object-Oriented Framework for Multimedia Composition and Synchronisation*, Object Composition (Centre Universitaire d' Informatique, Universitaite de Geneve, 1991), pp. 133-143.
2. T.Little, A. Ghafoor, *Spatio-temporal Composition of Multimedia*, IEEE Computer **24**(10, October 91), pp. 42-50.
3. S. Newcomb, N. Kipp, V. Newcomp, *THE HYTIME, Hypermedia Time based Document Structuring Language*, Communications of the ACM, **34**(11, November/91), pp. 67-83.
4. P. Hoepner, *Synchronizing the Presentation of Multimedia Objects*, ACM SIGOIS (January 1991), pp. 19-31.
5. R. Steinmetz, *Synchronisation Properties in Multimedia Systems*, IEEE Journal on Selected Areas in Communications **8**(3, April 1990), pp. 401-412.
6. J. F. Allen, *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM, **26** (11, November 1983)pp. 832-843.
7. Macromind director, *Interactivity Manual*, (Macromind Inc, 1990).
8. Y. Ishikawa, H. Tokuda, and C.W. Mercer, *Object-Oriented Real-Time Language Design: Constructs for Timing Constraints*, Technical Report CMU-CS-90-111 (Carnegie Mellon, March 1990).
9. S.T. Levi and A.K. Agrawala, *Real-Time System Design*, (McGraw-Hill, 1990).
10. J.A. Stankovic, *Misconceptions About Real-Time Computing - A Serious Problem for Next-Generation Systems*, Computer **21**(10, October 1988), pp.10-19.
11. C. Mourlas and C. Halatsis, *Extensions to a Parallel Prolog System to Support Real-Time Applications*, in Proceedings of the 4th International PARLE Conference (Paris 1992),pp. 551-565 .
12. ISO-IEC, *Multimedia and Hypermedia information coding Experts Group (MHEG)*, JTC1/SC29/WG12, (12/1991), pp. 70-72.
- 13.M. Vazirgiannis, C. Mourlas, *An object oriented model for interactive multimedia applications*, B.C.S.- Computer Journal **36**(1, January 1993).
14. V. de Mey, C. Breitener, L. Dami, S. Gibbs and D. Tsichritzis, *Visual Composition and Multimedia*, Object Frameworks, (Centre Universitaire d' Informatique, Universite de Geneve, 1992), pp 243-258.


```

(0 or e2,b4.button_up,crete_audio.play(),(,e2), on ^C
  crete_audio.stop()),
(3,b4.button_up, crete_image.display(dissolve),,,),
(b1.button_up,,PI1.execute(),,,),
(b2.button_up,,HI1.execute(),,,),
(b3.button_up,,TI1.execute(),,,),
(Close.button_up,,crete.stop(),,,)
>
);

Scenario script_PI1 (PI1,
  <0,back.button_up, , PI_1.execute(),,,on ^C generate e4>,
  <mv_play.button_up,
    (e4 or mv_stop.button_up or vd_pause.button_up,
anim.mv.play(),
  (emvst, emvend), , ,)
  (emvst or eauend, emvend, anim.aud.play(), (eaudst, eaudend), ,)
...
)
////////// application definition //////////
Application crete(<compl,PI1,TI1,TI2,HI1, TI3, TI4>, script_crete);

```