

EVENT AND ACTION REPRESENTATION AND COMPOSITION FOR MULTIMEDIA APPLICATION SCENARIO MODELLING.

M. Vazirgiannis, T. Sellis

Department of Electrical Engineering,
Division of Computer Science,
National Technical University of Athens

Abstract In this paper we present a model for the representation of multimedia applications based on the scenario concept. The scenario is described in terms of events and actions. The proposed model represents all the events that may occur in a multimedia application (originated from the system, the application or the user). As regards actions we propose a rich scheme for spatial and temporal composition representation based on operators that cover all the spatial and temporal relationships among multimedia objects. Finally the scenario of the application is represented as a set of scenario tuples that correspond to fundamental sets of actions originated by the same event (or events).

Keywords Interactive Multimedia Applications, Events, Temporal Composition, Scenario Modelling

1. Introduction

Multimedia applications can be very complex as regards the number of involved objects, transformations of the objects in the scope of an application and relationships among them. We regard a multimedia application as a container that includes scenes (aggregations of objects in a thematic domain) which in turn include objects that are transformed and interrelated for the purpose of the application. It is obvious that in a complex multimedia application it may be very difficult to describe all the possible functionality and paths the user or the application may follow. Thus an authors (who is usually a non-technical person) needs tools for high level but complete description of all aspects of a multimedia application.

A key issue in the representation of multimedia applications is the description of spatial and temporal composition of objects participating in the application. Moreover the relationships (synchronisation) among objects must be represented. Therefore, one can think of a multimedia application as an event based environment in which there is a rich set of events that may occur and define the flow of application. For instance, the end of a video sequence, the spatial coincidence of two objects in the application window, the occurrence of a pattern in a media object are events that may be exploited to trigger

other actions in an application. The events may further be composed in order to express richer and more complex conditions.

Complete representation of interactive multimedia applications is an open research issue. A multimedia application description consists of the following modules:

- the media objects that participate in the application along with the presentation specifications and transformations, and
- the description of the application functionality (i.e. the flow of the application and the response to occurring system, application or user events)

An important issue, that arises in the above context, is the composition of media in space and time. Current commercial systems do not adequately cover these issues especially those related to temporal synchronisation and communication between objects participating in the application using events. Therefore there is a need to study

- support for complex events: in an interactive multimedia application there may be situations that are highly complex in terms of events that are interrelated with respect to the functionality of the application.
- spatial composition, i.e. the topological relationships among multimedia objects as they are placed in an application window in the framework of a multimedia application.

In this paper we present a model for the representation of multimedia applications based on the scenario concept. The scenario is described in terms of events and actions. The events in a multimedia application cover a wide range of actions that may occur (originating from the system, the application, or the user). We derive a rich scheme for representing of simple and complex events regarding temporal, spatial and content based issues. We exploit ideas from the active databases area [GAT94] for modelling and composition of events. Moreover we define a scheme for representing actions in the framework of multimedia applications. The scheme supports composition of actions in the spatial and temporal domains. Finally, we present an integrated mechanism for description of application scenaria in terms of scenario tuples. A scenario tuple defines the application response to an event (simple or complex).

The proposed model is based on previous work on an object oriented framework that represents multimedia objects and applications [VAZ93][VAZ95]. This framework is extended towards the direction of

- the representation of simple and complex events
- the representation of spatial and temporal composition of multimedia objects, and
- more complete scenario definition.

2. Event modelling and classification

Events in the context of multimedia information systems widen the context of events, as used in the domain of active databases, since events convey either spatial and/or temporal information. Moreover the presentation of multimedia objects is a read-only action that has no delete or update actions to the multimedia database.

In an interactive multimedia application the various events that occur may be classified in external and internal. External events occur out of the application scope, are related to time and space and are due to changes happening independently of the application. External events may be time instances, time intervals, etc.

Internal events are raised from the application, either by objects participating in the application or by user events in the context of the application or by the system, and are functionally related to the application. Internal events are further classified into:

- state events: these events indicate values that denote the state an object is in (e.g. object A is active)
- condition events: In the context of an application we need to represent conditions that relate to the content of a media objects. These conditions raise events that may be consumed to trigger other actions. The condition may be of the form: (pattern, obj_id) where pattern is a pattern that is detected in the media object identified by obj_id.
- transitional: such events represent continuous changes in object properties or inter-object relationships (e.g. object A goes_away_from object B, audio A1 fades_out, image A1 wipes in image A2)
- method events: Method events relate to the behaviour of entities participating in the application and correspond to the execution of a method, i.e. some user-defined operation on an object.

2.1. Event Modelling

An initial hierarchy that represents the aforementioned classification of events appears in fig. 1. The following is the definition of the class that represents generic events

```
class event
    string          name
    object_id       subject
    object_id_list  object
    action_list     actions
end
```

The attribute name is the string indicating the semantics of the event. The attribute subject denotes the object that raises the event while the attribute object denotes the list of objects that are related to the event, and that are affected by the occurrence of the event. When there are no affected (related objects) a null value is attached. The attribute actions indicates the actions that raise this event. These actions may be class methods, conditions that are fulfilled etc.

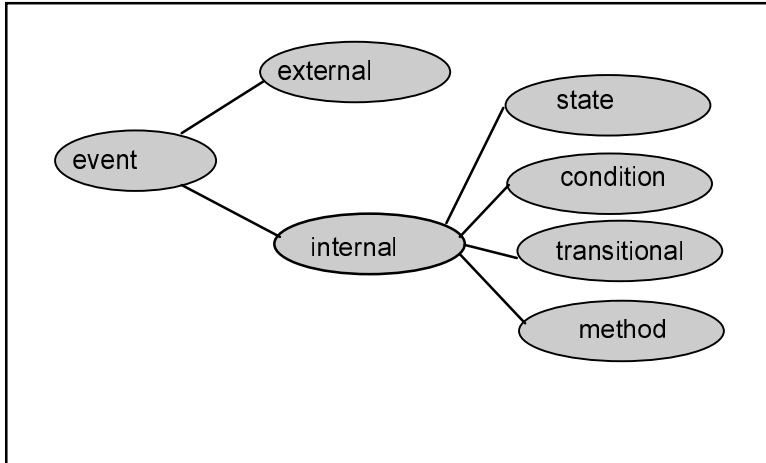


Fig. 1. An initial class hierarchy for representation of events

Examples of event definitions follow. Assume we want to represent the event *e1* that is being raised when the time instance 7:42pm comes. We assume that there is a class *timer* that represents time intervals and has the method *time_instance(t)* that returns true when value *t* comes. Then the event *e1*, according to the aforementioned class is as follows:

```

name      =    e1
subject   =    timer1
object    =    null
actions   =    time_instance(7:42pm)
  
```

Assume that we want to represent the event that object *A1* is active in an application window. According to [VAZ93] for an object that is active the method *get_status* returns TRUE. The corresponding state event instance (*A1isActive*) follows:

```

name      =    A1isActive
subject   =    A1
object    =    null
actions   =    (get_status = TRUE)
  
```

Another example demonstrating method events follows: assume we want to trap the invocation of the method *start()* when applied the video object *V1*. The corresponding event would be:

```

name      =    V1started
subject   =    V1
object    =    null
actions   =    start()
  
```

Finally the following event models the transitional event that is raised when object *A1* goes away from object *B1*:

name	=	A1goesawayB1
subject	=	A1
object	=	B1
actions	=	A1.increase_distance(B1)

2.2. Composition of Events

An important issue of interest is the ability to compose events in order to be able to represent complex events that are useful in real world applications. The composition of events as proposed in the current model is based on the operators proposed in [GAT94]. We propose the following operators for composition of events in the content of a multimedia application:

“;” conjunction of events (i.e. for two events A and B the composite event A;B occurs when both A and B have occurred).

“|” disjunction (i.e. for two events A and B the composite event A|B occurs when A or B has occurred)

“:t:” sequential occurrence (i.e. for two events A and B the composite event A:t:B occurs when both A and B have occurred and B has occurred t time units after A)

Moreover we define the following functions:

TIMES (n, e) (which means that the event e has occurred n consecutive times)

IN (e,int) (which means that the event e has occurred during the time interval int)

Based on the above, we can define the temporal interval (according to the EBNF syntax):

temporal interval = (start, end)

start = event | time

end = event | time

event = STRING | function

event = event {“|”, “;”, “:t:” event}

In active database systems there is a wider repertoire of functions and operators for events[GAT94]. However, in the case of multimedia applications the above set of operators and functions that manipulate events is considered sufficient.

3. Actions (Spatial and Temporal composition)

In a simple model of operation, the manipulation of multimedia objects is based on the actions that modify the spatial, temporal or storage features of them. Hoepner [HOE91] 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 consist of atomic or other composed actions

whose parts have to be synchronised. This classification is made mainly for synchronisation purposes so that atomic actions are used for synchronisation scenarios. The start and end points of an action are used as synchronisation points. Another classification of actions is attempted by [LIT91] 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.

Temporal Relationships

The topic of relations between temporal intervals has been addressed by [ALL83]. In that 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. In [VAZ93] a set of path operators that represent the aforementioned relationships are defined. This set defines the semantics of a synchronisation mechanism and the synchronisation of the presentation of multimedia objects. The operators which are also incorporated in the proposed model are: \wedge , \vee , $*$, $-t \rightarrow$. The semantics of the aforementioned operators are the following (see fig. 2.):

- The operators \vee and \wedge cover the relationships *starts* and *ends* respectively.
- $i*$: The execution of the action will be repeated i consecutive times.
- $A-t \rightarrow B$ where A and B are temporal intervals and t is a value that indicates the temporal interval between the end of interval A and the beginning of temporal interval B . This operator covers the relationships *before* (for $t > 0$), *meets* (for $t = 0$), *during* ($t < 0$ and $|t| > \text{duration } A$ and $\text{duration } B > \text{duration } A$), *overlaps_with* ($t < 0$ and $|t| < \text{duration } A$).

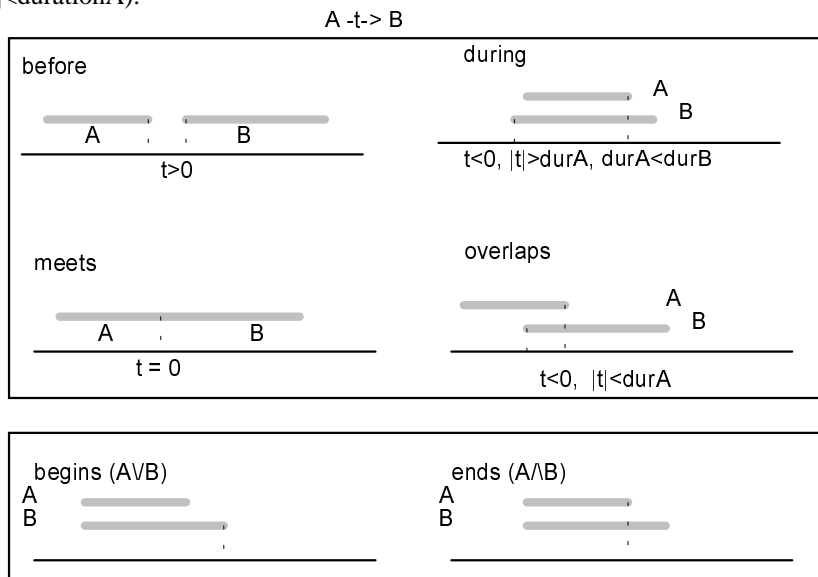


Fig. 2. The temporal operators that are defined and the relationships they represent

The formal definition of the operators based on the EBNF notation follow:

```
temporal operator
=      "∧"
|      "∨"
|      "-t"->"
|      "i*"
```

where t is a temporal interval as defined above, and i is an integer.

Spatial Composition

Another aspect of composition is the spatial one, regarding the spatial ordering and overlapping features of the participating objects. There is a complete set of topological relationships describing the spatial relationships between two objects [EGE91]. Thus two objects p, q (see fig. 2.) may coincide (equal), overlap, touch_externally (meet), touch_internally (covers), be inside, lie outside, or be disjoint .

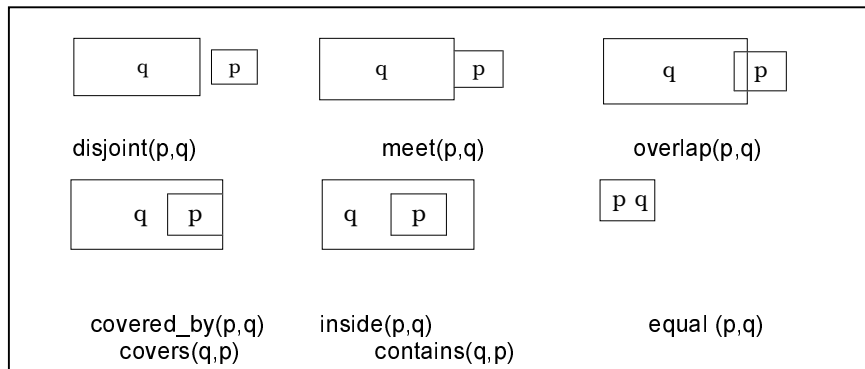


Fig. 3. Topological relations between two spatial objects

We define a set of binary operators that cover the topological relationships mentioned above

```
spatial operator
=      "disjoint"
|      "meet"
|      "overlap"
|      "covers"
|      "inside"
|      "equal"
```

The semantics of these operators appear in fig. 3.

4. Scenario Modelling

The scenario of an interactive multimedia application may be represented by a set of tuples indicating the functionality of particular actions. Scenario tuples can be represented by a class having the following attributes:

```
class scenario tuple
    event_expression      start_time
    event_expression      stop_time
    action_expression     action_list
    content_event_expression content_condition
    events                synch_events
    constraint_expression constraints
    boolean               is_active
end
```

start_time: It represents the event expression that triggers the execution of the actions described in the `action_list`. The attribute `stop_time` represents the event expression that terminates the execution of this tuple.

action_expression: Represents the list of actions that will take place and their composition in the spatial and temporal domains.

content_condition: an expression that indicates a condition that refers to the content of the multimedia objects included in this scenario tuple. The content condition must be true in order for the particular scenario tuple to be executed. If it is true then an event may be raised. Such an event may be exploited in the start and/or stop events expression; for example one minute after the appearance of the dog in the video start the execution of the tuple.

synch_events: refers to the events generated at the beginning and at the end of the current tuple execution. These events may be exploited for synchronisation.

constraints: this filter represents a set of constraints that must be fulfilled during the execution of the scenario tuple. These constraints refer to either the spatial or the temporal domain. The constraint expression is a spatial and/or temporal constraint that must be true so that the tuple is executed. For example, a constraint would be to start the execution of a tuple not after 13:00pm or to start the tuple only if the application window is big enough for the participating objects to be displayed without cropping.

is_active: this Boolean attribute indicates whether the tuple is active (i.e. the actions described in the tuple are in the execution phase) or not. When the tuple is activated (i.e. when the `start_time` expression becomes true and the actions in the `action_list` are executed) this attribute becomes TRUE. Otherwise it is FALSE. This attribute may be used for communication and condition checking purposes among tuples.

4.1. An example of a scenario

We assume a presentation for Western Crete regarding musical and tourist events. There is popular music from the Chania and the Rethimno districts as well as local events that are happening through out the year. The scenario that we would like to capture for such an application (see fig. 4.) is the following:

Starting the application, the images of the two districts are presented while a background music fades in. When the user moves the button FOOD an audio object starts playing (either the music CHANIA_MUS or the music RETHIMNON_MUS depending on which region is closer to the button). Moreover the music volume increases as the button approaches the district while the volume decreases when the button goes away from the district.

When the button EVENTS is entirely in the Chania region, a video starts playing (either the video SUMMER_EVENTS if system time is in the summer season or GENERAL_EVENTS in all other times). Pressing the button EXIT the user ends the application.

This multimedia presentation may be represented according to the proposed model. The representation consists of different modules referring to:

- the events that will be utilised by the application (i.e. start scenario_tuples execution etc.),
- the multimedia objects that participate in the multimedia application, along with their spatial and temporal transformations.
- the scenario tuples that represent the sets of actions that will be triggered by specific events or event expressions.

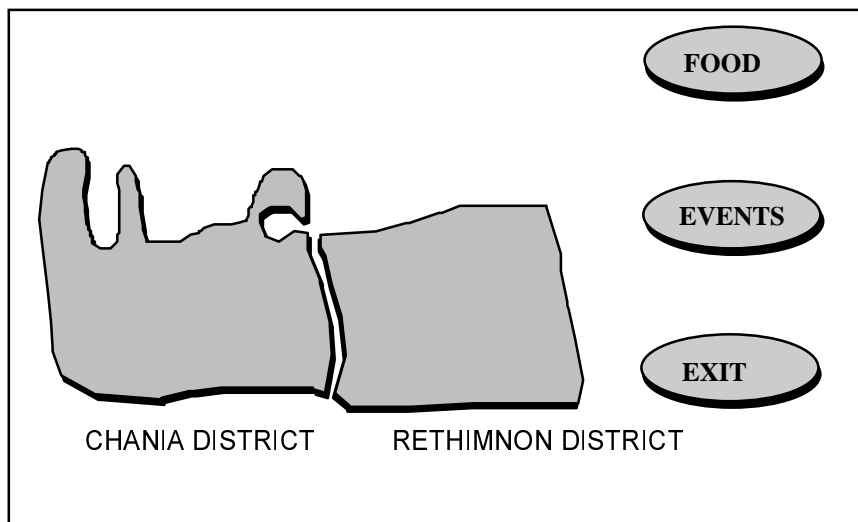


Fig. 4. The multimedia presentation of Western Crete.

The detailed description of the multimedia application follows:

events definition

In this module we represent the events that will trigger actions within the application. The events are the following:

F_M (this event is raised when the button FOOD is moved by the user):

```
name    = F_M
subject = b1
object  = NULL
action_list = b1.moving()
```

E_M (raised when the button EVENTS is moved by the user):

```
name    = E_M
subject = b2
object  = NULL
action_list = b2.moving()
```

E_P (raised when the button EXIT is pressed the user):

```
name    = E_P
subject = b3
object  = NULL
action_list = b1.button_down()
```

E_I_R (this event is raised when the button EVENTS is spatially in the area of object RETHIMNON_DISTRICT):

```
name    = E_I_R
subject = b2
object  = NULL
action_list = b2.inside(s2)
```

E_I_C (this event is raised when the button EVENTS is spatially in the area of object CHANIA_DISTRICT):

```
name    = E_I_C
subject = b2
object  = NULL
action_list = b2.inside(s1)
```

F_C_C (raised when the button FOOD approaches to CHANIA_DISTRICT):

```
name    = F_C_C
subject = b1
object  = s1
action_list = b1.distance(s1).decreases()
```

F_C_R (raised when the button FOOD approaches to RETHIMNON_DISTRICT).

```
name    = F_C_R
subject = b1
object  = s2
```

```
action_list = b1.distance(s2).decreases()
```

participating object definition:

We next define the objects participating in the application, according to the scheme presented in [VAZ95]. More specifically the media files, the temporal and/or spatial transformations and effects are described, and the spatial positions of the various objects in the application window are defined. According to [VAZ95] the multimedia objects are classified into temporal and spatial ones. The temporal are those that relate to time (sound, video) and spatial are those that relate to space(text, image, video). This classification is represented by the classes `t_object` and `s_objet` respectively. Class `t_object` represents the temporal features of a time dependent media object while the class `s_object` represents the spatial features of the media objects. In the case of a multimedia application we need to describe the participating media objects. However, in most of the cases the media objects are transformed (either spatially or temporally) in order to fulfil the application requirements. In the model presented in [VAZ95], these transformations are represented by the classes `t_media_tuple`, `s_media_tuple`. In the same paper the user interface element button is represented by the respective class as a subclass of class `s_object`. The objects that participate in the sample application are the following:

- *temporal objects*

The description of the temporal (i.e. audio) objects, along with their temporal transformations, that participate in the scene is following:

a1 (audio object described as background music):

```
t_media_tuple a1
attributes
    m_obj          = "BACKGROUND MUSIC"
    t_scale_f      = 1
    t_start        = 0
    t_duration     = null
    direction      = TRUE
    effect_id      = fade_in
end
```

a2 (audio object described as CHANIA_MUSIC music):

```
t_media_tuple a2
attributes
    m_obj          = "CHANIA_MUSIC"
    t_scale_f      = 1
    t_start        = 0
    t_duration     = null
    direction      = TRUE
    effect_id      = null
end
```

a3 (audio object described as RETHIMNON_MUSIC music).

```

t_media_tuple a3
attributes
    m_obj          = "RETHIMNON_MUSIC"
    t_scale_f      = 1
    t_start        = 0
    t_duration     = null
    direction      = TRUE
    effect_id      = null
end

```

- *spatial objects*

The description of the spatial (i.e. text, image, video) objects along with their spatial transformations that participate in the scene is following:

s1 (the image for the Chania district):

```

s_media_tuple s1
attributes
    m_obj          = "CHANIA_DISTRICT.bmp"
    t_scale_f      = null
    t_start        = null
    t_duration     = null
    direction      = null
    effect_id      = 0
    x_coord        = 50
    y_coord        = 200
    a_width        =
    a_height       =
    effect_id      = 0
    s_scale_f      = 1
end

```

s2 (the image for the Rethimnon district):

```

s_media_tuple s2
attributes
    m_obj          = "RETHIMNON_DISTRICT.bmp"
    t_scale_f      = null
    t_start        = null
    t_duration     = null
    direction      = null
    effect_id      = 0
    x_coord        = 350
    y_coord        = 200
    a_width        =
    a_height       =
    effect_id      = 0
    s_scale_f      = 1
end

```

s3 (the video for SUMMER_EVENTS):

```

s_media_tuple s3
attributes

```

```

    m_obj      = "SUMMER_EVENTS.avi"
    t_scale_f  = null
    t_start    = 0
    t_duration = 15
    direction  = TRUE
    effect_id  = 0
    x_coord    = 200
    y_coord    = 200
    a_width    =
    a_height   =
    effect_id  = 0
    s_scale_f  = 1
end

```

s4 (the video for GENERAL_EVENTS):

```

s_media_tuple s4
attributes
    m_obj      = "GENERAL_EVENTS.avi"
    t_scale_f  = 1
    t_start    = 0
    t_duration = 20
    direction  = TRUE
    effect_id  = 0
    x_coord    = 200
    y_coord    = 200
    a_width    =
    a_height   =
    effect_id  = 0
    s_scale_f  = 1
end

```

b1 (for button FOOD):

```

button B1
attributes
    text      = "FOOD"
    state     = FALSE
    enabled   = FALSE
    object_time = 0
    t_length  = 0
    t_status  = 0
    t_scale_f = 1.0
    looping   = FALSE
    direction = TRUE
    m_width   = 100
    m_height  = 50
    plane     = 0
    transparency = 0
    focus     = FALSE
end

```

b2 (for button EVENTS):

```
button B2
attributes
    text          = "EVENTS"
    state         = FALSE
    enabled       = FALSE
    object_time   = 0
    t_length      = 0
    t_status      = 0
    t_scale_f     = 1.0
    looping       = FALSE
    direction     = TRUE
    m_width       = 100
    m_height      = 50
    plane         = 0
    transparency  = 0
    focus        = FALSE
end
```

b3 (for button EXIT).

```
button B3
attributes
    text          = "EXIT"
    state         = FALSE
    enabled       = FALSE
    object_time   = 0
    t_length      = 0
    t_status      = 0
    t_scale_f     = 1.0
    looping       = FALSE
    direction     = TRUE
    m_width       = 100
    m_height      = 50
    plane         = 0
    transparency  = 0
    focus        = FALSE
end
```

- *scenario definition*

We now define the application functionality in terms of scenario tuples that define the flow of the application in terms of events, actions and conditions. The first tuple (T1) starts at the start of the application (application_time = 0) and causes display of the CHANIA_DISTRICT and RETHIMNON_DISTRICT images as well playback of the background music:

```
scenario tuple T1
    start_time    = 0
    stop_time     = E_P
    action_list   = s1.execute() ^ s2.execute ^ s3.execute()
    content_condition = null
    synch_events  = (.,_)
    constraints   = null
```

```

is_active      =
end

```

The tuple T2(T3) starts playing the music CHANIA_MUS (RETHIMNON_MUS) when the button FOOD starts moving and is coming closer to CHANIA_DISTRICT (RETHIMNON_DISTRICT) than to RETHIMNON_DISTRICT (CHANIA_DISTRICT):

```

scenario tuple  T2
  start_time    =      F_M | (e22 ; F_M)
  stop_time     =      (T3.get_active() = true) | not (F_M) | E_P
  action_list   =      t2.execute()
  content_condition = null
  synch_events  =      (_, e22)
  constraints    =      null
  is_active     =
end

```

```

scenario tuple  T3
  start_time    =      F_M | (e23 ; F_M)
  stop_time     =      (T2.get_active() = true) | not (F_M) | E_P
  action_list   =      t3.execute()
  content_condition = null
  synch_events  =      (_, e23)
  constraints    =      null
  is_active     =
end

```

Tuple T4 starts when button EVENTS is entirely in the CHANIA_DISTRICT. Then the video SUMMER_EVENTS starts. The condition for this tuple to be executed is that the system time is in the summer period:

```

scenario tuple  T4
  start_time    =      E_I_C
  stop_time     =      E_P
  action_list   =      s3.execute()
  content_condition = null
  synch_events  =      (_, _)
  constraints    =      IN(sys_time, summer)
  is_active     =
end

```

Tuple T5 starts when button EVENTS is entirely in the CHANIA_DISTRICT. Then the video GENERAL_EVENTS starts. The condition for this tuple to be executed is that the system time is in not the summer period.

```

scenario tuple  T5
  start_time    =      E_I_C
  stop_time     =      E_P
  action_list   =      s4.execute()
  content_condition = null
  synch_events  =      (_, _)

```

```

        constraints = NOT(IN(sys_time, summer))
        is_active =
end

```

Tuple T6 starts when button EXIT is pressed. The execution of this tuple causes the end of the application. The event E_P, described above, triggers the end of all actions in all the tuples that are active.

```

scenario tuple T6
    start_time = E_P
    stop_time =
    action_list = scene.exit()
    content_condition = null
    synch_events = (_, _)
    constraints = null
    is_active =
end

```

Tuple T7 increases the volume of CHANIA_MUSIC as the button FOOD approaches to CHANIA district. The tuple is executed if tuple T2 is active (i.e. the CHANIA_MUSIC is being played back). Tuple T8 increases the volume of RETHIMNON_MUSIC as the button FOOD approaches to RETHIMNON district. The tuple is executed if tuple T3 is active (i.e. the RETHIMNON_MUSIC is played back).

```

scenario tuple T7
    start_time = F_M
    stop_time = E_P
    action_list = t2.increase_volume()
    content_condition = null
    synch_events = (_, _)
    constraints = T2.get_active()==true
    is_active =
end

```

```

scenario tuple T8
    start_time = F_M
    stop_time = E_P
    action_list = t3.increase_volume()
    content_condition = null
    synch_events = (_, _)
    constraints = T3.get_active()==true
    is_active =
end

```

5. Related Work

Work in the area of active databases and multimedia application modelling is relevant to the work presented here. As regards active databases, we exploited ideas from efforts that aim at modelling and manipulation of events. Previous work that models events and their composition is described in [GAT94]. This model is mainly related to active databases and classifies events in primitive and composite ones. Primitive events are further classified into Time, Method and Abstract events. Composite events are

composed of primitive events according to a set of operators (“;” conjunction, “|” disjunction, “:” sequential occurrence) and functions [GAT94].

As regards multimedia composition and synchronisation modelling there is a variety of approaches. In [LIT93] a model is presented for the representation and manipulation of relationships among temporal intervals. It introduces the concepts of temporal instant and interval and the actions of temporal access control: start, stop, fast forward, rewind, pause, resume. The innovative features of the proposed model are that it defines a set of n-ary operators for the representation of relationships among many temporal intervals. Moreover the inverse relationships are defined (binary and n-ary). Finally an algorithm for the evaluation of a temporal expression is introduced for calculation of the overall duration, and the time of the temporal composition. This model does not address the scenario concept at all. Moreover it is rather a temporal composition model rather than multimedia application representation scheme.

Another alternative view of synchronisation models is based on Petri Nets. The basic idea in these models is to represent various components of multimedia objects as places and describe their inter-relationships in the form of transitions. Timed Petri Nets have been extended to develop a model that is known as Object Composition Petri Nets [LIT93]. The particularly interesting features of this model are the ability to explicitly capture all the necessary temporal relations and to provide simulation of presentation in both forward and reverse directions. An OCPN can represent all thirteen possible temporal relations between two temporal intervals.

In another model, called Petri-Net-Based-Hypertext (PNBH), higher level browsing semantics can be specified. In this model, information units are treated as net places and links as net arcs. Transitions in a PNBH indicate traversal of a link or the browsing of information fragments. The segments of a PNBH can be played-out in random order, as selected by the user and restricted by the semantics of the net. The disadvantage of the Petri-net based models is that there is no mechanism to specify communication requirements and control functions for distributed composition objects.

Finally, another kind of models are the Object Oriented ones. The basic idea in this approach is to represent a real world thing or concept as an object. An object usually has an identifier, attributes, methods, a pointer to data etc. One such approach is the OMEGA system [CHA94]. In this model a multimedia object has attributes, relationships which are its value reference to other objects, components and methods. To facilitate the presentation of multimedia objects OMEGA uses temporal information associated with each object to calculate precedence and synchronisation between objects.

6. Conclusions and Research Directions

The approach presented in this paper is a first attempt for the description of complex multimedia applications. The formalism proposed is declarative thus facilitating shorter descriptions. More specifically the proposed model defines a framework in which

- complex events,
- complex actions,
- applications scenaria

may be defined in a declarative way.

There are several issues that need to be further investigated in the framework of this model:

Complete set of events

We need to investigate the design of a complete repertoire of events in all the categories mentioned in Section 2. The effort is especially difficult in the area of condition events. As regards transitional events, rich formalisms should be adopted in order to describe transition in a multimedia application adequately.

Spatial and temporal constraint checking at specification time

The scenario tuples indicate actions that will be executed when events occur. Event composition descriptions may not always be consistent with the application state or the system resources, and therefore mechanisms need to be devised for consistency checking.

Scenario execution

An important issue is how scenarios are executed. This is a complex issue since it involves resource allocation, disk management, device characteristics etc. It is to some extent an orthogonal issue to composition specification, but for realistic implementations it has to be taken into account.

References

[ALL83]. J. F. Allen: Maintaining Knowledge about Temporal Intervals, *Communications of the ACM*, Vol. 26, No. 11, November 1983, pp. 832-843.

[CHA94] A. Ghafoor (1994): Multimedia Databases Coursenotes, *ACM Multimedia 94 Conference*.

[EGE91] Egenhofer, M., Franzosa R: Point-Set Topological Spatial Relations, *International Journal of Geographic Information Systems*, Vol 5, No. 2, pp. 160-174.

[GAT94] S. Gatzju, K. Dittrich: Detecting Composite Events in Active Database Systems Using PetriNets, *Proc. of 4th Intl. Workshop on Research Issues in Data Engineering: Active database systems*, Texas, February, 1994

[HOE91] P. Hoepner: Synchronizing the Presentation of Multimedia Objects, *ACM SIGOIS*, January 1991, pp. 19-31.

[LIT91] T.Little, A. Ghafoor: Spatio-temporal Composition of Multimedia, *IEEE Computer*, Vol24, No. 10, October 91, pp. 42-50.

[LIT93] T.Little, A. Ghafoor: Interval-Based Conceptual Models for Tim-Dependent Multimedia Data, *IEEE Transactions on Data and Knowledge Engineering*, Vol. 5, No. 4, pp. 551-563, August 93.

[VAZ93] M. Vazirgiannis, M. Hatzopoulos: A Script Based Approach for Interactive Multimedia Applications, *Proceedings of the MMM (International Conference on Multi-Media Modelling) Conference*, Singapore, (November 9-12, 1993).

[VAZ95] M. Vazirgiannis, M. Hatzopoulos: Integrated Multimedia Object And Application Modelling based on events and scenarios, *Proceedings of 1st IEEE International Workshop for MMDBMSs*, 8/1995, Blue Mountain Lake, NY (The Adirondacks).