Interactive Multimedia Documents:

a Modeling, Authoring and Rendering approach

M. Vazirgiannis

Department of Informatics,
Athens Economic &
Bussiness University,
Patision 76, 10434, Athens,
HELLAS

D. Tsirikos, Th.Markousis, M. Trafalis, Y. Stamati, M. Hatzopoulos

Department of Informatics, University of Athens,
TYPA buildings, University Campus,
15771 Ilisia, Athens,
HELLAS

T. Sellis

Computer Science Division,
Department of Electrical and
Computer Engineering,
National Technical University
of Athens, 15773 Zographou,
Athens,
HELLAS

Abstract

In this paper we present our research and development experience in the context of Interactive Multimedia Documents (IMDs). We define a rich model for such documents covering the issues of interaction and spatiotemporal compositions as means of representing the functionality of an IMD. We exploit the event concept to represent interaction, while complex interactions are covered by a rich algebraic and spatiotemporal event composition scheme. Based on the model we implemented an authoring methodology. Thereafter we present a generic framework for rendering (presenting) IMDs putting emphasis to the handling of interaction and to the temporal synchronization of media objects. The rendering system is implemented as a client-server architecture using Java and accompanying technologies. The implementation is suitable for WWW enabled interactive multimedia documents.

1 Introduction

An Interactive Multimedia Document (IMD) involves a variety of individual multimedia objects presented according to a set of specifications called the IMD scenario. These multimedia objects are transformed (spatially and/or temporally) in order to be presented according to author's requirements. The author also defines the way the user will interact with the presentation session as well as the way the application will treat application or system events. The related application domains are quite challenging and demanding. Possible candidates include: interactive TV, digital movies and virtual reality applications. In the framework of IMDs we consider the following as cornerstone concepts:

- *Actors*: they represent the media objects participating (video, sound, image, text, and buttons) and their spatiotemporal transformations [V96].
- *Events*: they are the fundamental means of interaction in the context of the IMD and are raised by user actions, by objects participating in the IMD or by the system. Events may be simple (i.e. not decomposable in the IMD context) or complex, and have attached their spatiotemporal signature (i.e. the space and the time they occurred). For more details refer to [VB97].
- *Spatiotemporal Composition*: it is an essential part of an IMD and represents the spatial and temporal ordering of media objects in the corresponding domain. At this point, the issue of spatial and temporal relationships among the objects is critical [VTS98].
- *Scenario*: it stands for the integrated behavioral contents of the IMD, i.e. what kind of events the IMD will consume and what presentation actions will be triggered as a result. In our approach a scenario consists of a set of self-standing functional units (scenario *tuples*) that include: triggering events (for start and stop), presentation actions (in terms of spatiotemporal compositions called *instruction streams*) to be carried out in the context of the scenario tuple, and related synchronization events (i.e. events that get triggered when a scenario tuple starts or stops). Instruction streams are expressions that involve Temporal Access Control (TAC) actions such as start, stop, pause, resume and others, on actors with the use of vacant temporal intervals in between. In [VTS98] a set of operators has been defined for the TAC actions and for the corresponding events. Thus, the expression (A>3B>0C!) is interpreted as “start A, after 3 seconds start B and immediately after (0 seconds) stop C”.
- Scenario Rendering: it is the process of defining an execution plan for the IMD scenario, i.e. when, where, for how long and under which transformations each media object will be presented. This task is rather straightforward in “pre-orchestrated” applications [KE96], while in the case of IMDs with rich interaction it becomes more complicated.

In this research effort we define a model that widely covers the issue of interactive scenarios and their constituents: events and spatiotemporal composition of presentation actions. According to our model, an

IMD is defined in terms of *actors*, *events* and *scenario tuples*. The result of IMD specification under this approach is a declarative script.

Based on the model we developed an authoring methodology, consisting of three phases: i. selection and transformation of the media objects to participate ii. definition of the events (atomic and complex) that the IMD session will consume; the author defines the atomic events that the current IMD will exploit for the scenario, and iii. specification of the scenario of the IMD session in terms of scenario tuples (autonomous functionality units relating interactions with presentation actions)

As already mentioned, the rendering of an IMD scenario may be a complicated issue regarding the multitude of occurring events and the potentially large sets of instruction streams. The rendering of the scenario is based on a generic multi-threaded architecture that detects and evaluates events that occur in the presentation context and triggers the appropriate synchronized presentation actions. The rendering scheme has been implemented in a client-server system based on Java. The server provides the IMD, while the media objects reside in any http server. The client, in charge of rendering the IMD scenario, retrieves the scenario from the server and requests the appropriate media from the corresponding http servers. The client design widely covers the issue of interaction with external and internal entities in terms of simple and complex events. The system detects and evaluates the occurring events and triggers the appropriate presentation actions. Furthermore, it maintains the high-level spatial and temporal synchronization requirements of the application according to the IMD scenario. The whole framework has been implemented in Java using the RMI (Remote Method Invocation) client-server communication protocol and the JMF (Java Media Framework) for handling multimedia objects. The system presents a promising approach for distributed interactive multimedia on the Internet and Intranets.

The paper is organized in the following way: in section 2 we demonstrate the theoretical model for IMD scenarios. More specifically in section 2.1 we present a classification scheme for events and a model that represents events as carriers of interaction in the context of an IMD. The model covers atomic and composite events, while the composition scheme (section 2.2) covers algebraic and spatiotemporal aspects. In section 3, we present the IMD scenario model in terms of scenario tuples representing autonomous interaction-presentation entities. Section 4 is devoted to implementation experiences while putting the theoretical model into practice. At first (section 4.1) we propose an authoring methodology. Following (section 4.2), we present a generic IMD scenario document-rendering scheme. In this scheme the scenario tuples are triggered/stopped according to the events occurring in an IMD session. More specifically, we analyze a generic detection and evaluation scheme for events in an IMD session. In section 5 we discuss related work in the areas of Multimedia Document standards, IMD rendering and in the related issue of the active databases domain (event detection & evaluation and rule triggering schemes). We conclude in section 6, by summarizing and discussing further research and development directions.

2 The IMD Scenario model

To support complex IMDs, a system that offers both a suitable high-level modeling of IMDs and interactive multimedia presentation capabilities is needed. The modeling should cater for the spatial and temporal composition of the participating media, the definition of interaction between the user and the IMD, and the specification of media synchronization.

There are many approaches to model pre-orchestrated multimedia scenarios [SKD96, WRW95, WWR95, ISO93, LG93, ISO92]. The proposed models are mainly concerned with temporal aspects and synchronization. The support of user interaction in these models, however, is often lacking or unsatisfactory. In the multimedia literature, interaction is hardly addressed as a research issue. Moreover, the few approaches mentioned above that take into account interaction [SKD96, WRW95, WWR95, ISO93] are very limited, since they refer mostly to simple choices, jumps, selections and temporal control actions (start, stop, pause, resume etc.) offered to the user by means of buttons, menus and sliders. One of the important aspects in the modeling of IMDs is the spatiotemporal composition to relate multiple media in the temporal and spatial dimensions. There are several approaches to model the temporal aspects of IMDs [LG93,WR94, BZ93], while the spatial aspects are rather under-addressed in the literature. Some of the interesting efforts in this area are described in [VTS96, IDG94].

We claim that modeling of IMDs should give more emphasis to the interactive parts of such an application. In principle, the modeling of interaction should cover all the procedures that somehow involve the machine and the user. Moreover, interaction is not limited to the reactions between the user and the computer, but can also take place between entities within the computer. For instance, objects that participate in an IMD interact spatially and/or temporally with each other, e.g. if two moving images meet across the screen. Also,

resources or devices produce messages/signals that may trigger actions in the context of the IMD, e.g. if no audio device is available the equivalent text representation is displayed. When presenting multimedia scenarios, all occurring interactions must be sensed and reacted to adequately as fast as possible.

Operations on multimedia objects imply actions that modify the spatial or temporal dimensions of these objects. *Actions* describe parts of the temporal and spatial course of an IMD in contrast to *interactions* that define the course of an IMD. Actions can be either fundamental or composite. A fundamental (or atomic) action is, for example, "start a video", "present or hide an image", etc. Composite actions are composed of atomic actions in the spatial and the temporal dimensions, according to a set of appropriate operators. For instance: "start video clip A, *and* 3 seconds after that, start audio clip B". In our approach we make use of this action notion, which is defined in more detail in [VTS96, VH93], in order to allow the modeling of multimedia applications to cover complex interactions.

2.1 Modeling interaction with events

The concept of events is defined in several research areas. In the area of Active Databases [G94, GJS92, CM93] an event is defined as an instantaneous *happening of interest* [GJS92]. An event is caused by some action that happens at a specific point in time and may be atomic or composite. In the multimedia literature, events are not uniformly defined. In [SW95] events are defined as a temporal composition of objects, thus they have a temporal duration. In other proposals for multimedia composition and presentation ([HFK95, VS96]), events correspond to temporal instances. We follow the latter understanding of the temporal aspect of events and consider events to be instantaneous.

Multimedia information systems, however, widen the context of events, as defined in the domain of active databases. In addition to the temporal aspect of an event, which is represented by a *temporal instance*, there are events in IMDs that convey spatial information. This is represented by a *spatial instance*. For example, an event captures the position of visual objects at a certain point in time. Another aspect that is also crucial in the IMDs context is that, although the number and multitude of events that are produced both by the user and the system may be huge, we may be interested only in a small subset of them. Thus, events must be treated in a different way in this context, as compared to Active Databases.

We define an event in the context of IMDs as follows:

> ***An event is raised by the occurrence of an action and has attached a spatial and temporal instance. The event is recognized by some interested human or process.***

As mentioned in the previous definition, all events have attached to them a temporal instance relative to some reference point, usually the beginning of the IMD. Apart from a temporal instance we assign a spatial instance to an event in case it is related to a visual media object. This spatial instance is essentially the rectangle that bounds an event (e.g., the screen area where the presentation of an image takes place). In some trivial cases, though (e.g., mouse click), the rectangle is reduced to a point.

Thus, it is meaningful to integrate the two notions of temporal and spatial instances in the definition of events. Therefore, we introduce the term *spatiotemporal instance* whose representation in tuple form is: (sp_inst, temp_inst), where sp_inst is a spatial instance and temp_inst is a temporal instance as defined in sections 2.1 and 2.2, respectively. However, events can be purely temporal, as this is the case for the start event of an audio clip.

2.1.1 Classification

In order to assist the authors in the specification of IMDs, we have to provide them with a fundamental *repertoire* of events. In the framework of IMDs we further classify the events into categories. The classification of the events is done on the basis of the entity that produces the event. The elaborated categories are those presented in further sections. This classification forms the basis for the object-oriented modeling, which we propose in section 3

user interaction

These are the events that are generated explicitly by user interactions within the IMD context. They are input events as the user interacts with the system via input devices such as mouse, keyboard, touch screen, etc. Other input events categories also exist, but they may not be applicable for the moment. However we refer to them for the completeness of the approach; such events are, for instance, voice input events and events related to touch-screens.

Temporal access control events are the well known actions *start, pause, resume, stop, fast forward, rewind,* and *random positioning in time* and concern the execution of one or a group of media objects. They bear, however, specific semantics for the manipulation of a media object's presentation initiated by the user, and this is the reason why we consider them separately in the classification procedure.

Intra-object events

This category includes events that are related to the internal functionality of an object presented in an IMD. This functionality is implemented in object-oriented approaches as method invocation. For instance, the invocation of a method corresponding to temporal access control such as myObject.start(), produces an intra-object event. Another source of intra-object events is state changes. The viable states of an object as regards its presentation may be temporal (*active, idle, suspended*) and/or spatial (*shown, hidden,* and *layer classification* information, etc.). State events occur when there is a state change of a media object, e.g., image I gets hidden, audio A is started, etc. Intra-object events may indicate a discontinuity in the continuous presentation of a media object.

Inter-object events

Such events occur when two or more objects are involved in the *occurrence of an action of interest*. These events are raised if spatial and/or temporal relationships between two or more objects hold. In the spatial case, an inter-object event can occur if one object, moving spatially, meets another media object. A temporal inter-object event can occur when the deviation between the synchronized presentation of two continuous media objects exceeds a threshold. Moreover, we may consider spatiotemporal inter-media synchronization events. Such events occur, for instance, when two objects are in relative motion during a specified temporal interval (e.g., object A approaches object B before 2am). Although system signals are not classified separately, they can result in inter-object events if, e.g., a low network capacity leads to a video presentation that is too slow (intra-object event) and this raises an inter-object event as the video presentation and an associated audio presentation are not synchronized anymore.

Application events

In this category we are interested in events related to the IMD *state*. An IMD state bears information that has been contributed essentially by all the objects currently presented in the IMD. Indeed, an IMD as a whole can be *idle*, *suspended*, or *active*. Moreover, during the execution of an IMD, overall Quality of Service (QoS) parameters must to be taken into consideration [TKWPC96]. An application event can also indicate that the overall QoS falls below a given (user defined) threshold. For instance, an IMD presents synchronized video and audio data and the network capacity is low. Then, single audio and video presentation generates intra-object events that indicate that their QoS is deteriorating, so as to initiate corresponding actions on the video and the audio presentation. Additionally, timer events may be of interest to denote certain points on the timeline. System signals that indicate the actual system state (such as low network capacity) result in application events that (e.g. indicate that the video presentation is too slow) are not classified separately.

User-defined events

In this category we classify the events that are defined by the IMD designer. They are related to the content of the IMD execution. A user-defined event can refer to the content of a media object, i.e. to the occurrence of a specific pattern in a media object. For instance, an event is to be raised if the head of a news speaker occurs in a video frame to indicate that the boring advertisements are over and the interesting news are now on.

IMD authors and developers are provided with an initial set of atomic events, namely the ones in the aforementioned categories. In many cases, though, they need to define complex events that are somehow composed of the atomic ones. For instance, assume two atomic events e_1 and e_2 corresponding to the start of video clips A and B, respectively. The video clips are started due to events previously occurred in the IMD. The author might define a composite event e_3 to be raised when e_1 occurred within 3 seconds after e_2 occurred. Another example of a composite event is the occurrence of temporally overlapping presentations of two images img_1 and img_2 between 10:00am and 11:00am. Thus, it is desirable that composite events may be defined by means of a provided set of composition operators. The necessary operators are defined and classified in further sections.Hereafter we present in summary a model for simple and complex events in IMDs, based on the event concept and classification that has been presented above. A detailed presentation can be found in [VB97].

2.1.2 Modeling and composition of events

According to the aforementioned event definition, we need the following attributes to represent a generic event: The subject and object attributes, that are of type objectList essentially representing the objects that caused or are affected by the event, respectively. The attribute spatio_temporal_signature that takes to the spatial and temporal instances attached to the event when it actually occurs. Then, the structure of the Event class in an object-oriented pseudo language is as follows:

```
class Event inherits from object
attributes // attribute name                    attribute's data type,
            subject                                     objectList;
            action                                      actionList;
            object                                      objectList;
            spatio_temporal_signature           spatiotemp_instance;
end
```

As we mentioned before, it is important to provide the tools to the authors for the definition of composite events. The composition of events in the context of an IMD has two aspects:

- *Algebraic composition* is the composition of events according to algebraic operators, adapted to the needs and features of an IMD.
- *Spatiotemporal composition* reflects the spatial and temporal relationships between events.

Before explaining these two aspects, we should define some fundamental concepts:

Spatiotemporal reference point **(θ)*:*** it is the spatiotemporal start of the IMD scenario named as θ. This serves as the reference point for every spatiotemporal event and instance in the IMD.
Temporal interval: it is the temporal distance between two events (e_1, e_2) namely the start and end of the interval t_int :== (e_1,e_2), where e_1,e_2 are events that may either be attached to predefined temporal instances relative to some reference or occur asynchronously.

Algebraic Composition of Events

In many cases the author wants to define specific events that relate to other existing events. We exploited some of the operators presented in other proposals on composite events in Active Databases [G94, GJS92]. We distinguish between the following cases:

Disjunction:
e :== OR(e_1,…,e_n): This event occurs when at least one of the events e_1,…,e_n occurs. For instance, we may be interested in the event e occurring when button A (e_1) or button B (e_2) was pressed.

Conjunction:
e :== ANY(k,e_1,…,e_n): This event occurs when at least any k of the events e_1,…,e_n occur. The sequence of occurrence is irrelevant. For example, in an interactive game a user proceeds to the next level when she/he is successful in two out of three tests that generate the corresponding events e_1,e_2,and e_3.

e :== SEQ(e_1,…,e_n): This event occurs when all events e_1,…,e_n occur in the order appearing in the list. For example, in another interactive game the user proceeds to the next level when she/he succeeds in three tests causing the events e_1,e_2,and e_3 one after the other.

e :== TIMES (n,e_1): This event occurs when there are n consecutive occurrences of event e_1. This implies that other events *may* occur in-between the occurrences of e_1.

In many cases the authors want to apply constraints related to event occurrences in specific temporal intervals. To facilitate this requirement we define a set of operators that are of interest in the context of multimedia applications:

Inclusion:
e :== IN(e_1,t_int), event e occurs when event e_1 occurs during the temporal interval t_int. For example, in an IMD we might want to detect three mouse clicks in an interval of 1 sec., so that a help window appears. If t_int = (e_2,e_3), where e_2 corresponds to the starting point of a timer while e_3 corresponds to the end of a timer whose duration is defined as 1 second. The desired event would then be defined as e=IN (TIMES(3,mouse.click), t_int).

Negation:
e :== NOT(e_1,t_int): event e occurs when e_1 does not occur during the temporal interval t_int.

***Strictly consecutive events*:**
In some cases we are interested in whether a series of events of interest is "pure" or mixed up with other events occurring. The event e:== S_CON(e_1,…,e_n) is raised when all of e_1,…,e_n have occurred in the order appearing in the list and *no other* event occurred in between them.

Apart from the above set of operators we also support the three known Boolean operators, namely AND, OR and NOT that have their usual meaning.

Spatiotemporal composition of events

In an IMD Session many events may occur. Each of them bears its spatiotemporal signature as described in previous sections. In many cases we are interested to recognize spatiotemporal sequences of events. The term spatiotemporal sequence stands for the spatial and/or temporal ordering of events, e.g., event e1 to occur spatially below and/or temporally after e2.

Temporal composition of events

Hereby we discuss the temporal aspect. Since events are of zero temporal duration, the only valid temporal relationships between two events are *after, before,* and *simultaneously* defined as follows:

After: e :== e_1 after e_2, event e occurs when e_1,e_2 have both occurred and that e_1 occurred before e_2.
Before: e :== e_1 before e_2, event e occurs when e_1,e_2 have both occurred and that e_2 occurred before e_1.
Simultaneously: e :== e_1 simultaneously e_2, event e occurs when e_1,e_2 have occurred at the same time.
There are cases in which algebraic operators and temporal relationships may be used interchangeably. For instance, for two events e_1,e_2 the expressions after(e_1,e_2) and SEQ(e_1,e_2) convey the same fundamental semantics. Although the former is a temporal relationship between *two* events, the latter is a conjunction operator bearing also temporal semantics for a *list* of events.

Spatial composition of events

The spatial aspects of events' compositions are related to position and/or motion of spatial objects (images, buttons etc.). We can assume that each event has a spatial signature that is the rectangle (Minimum Bounding Rectangle) that bounds the area of the event. Spatial relationships between objects involve three different aspects: topology, direction and metrics. A model that represents all these aspects is defined in [VTS96]. We will exploit this model for defining a complete spatiotemporal event composition scheme.

Spatiotemporal composition of events

Having defined all the temporal and spatial relationships among events, we can now introduce a complete spatiotemporal composition scheme. This set of relationships includes the set of all possible spatiotemporal relationships (169 spatial relationships * 3 temporal relationships = 507 spatiotemporal relationships) between two events.

A requirement that arises during event composition specification, is the definition of metrics between events. Thus, we define the notion of *spatiotemporal distance* between two events. For this purpose we consider the definition of the spatial distance between two rectangles as the Euclidean distance between their *closest vertices* as defined in [VTS96]. For our spatiotemporal composition scheme we extend this definition with the temporal distance concept, resulting in the following definition:

Spatiotemporal distance:
Given two events e_1, e_2 having corresponding spatiotemporal signatures: (x_{11}, y_{11}, x_{12}, y_{12}, t_1), (x_{21}, y_{21}, x_{22}, y_{12}, t_2), then the spatiotemporal distance of the two events is defined as: spatio_temporal_distance:== $sqrt((x_a - x_b)^2 + (y_a - y_b)^2 + (t_2 - t_1)^2)$, where ($x_a$, y_a), (x_b, y_b) are the coordinates of the closest vertices of the two spatial instances.

The generic spatiotemporal composition scheme for events, based on the EBNF notation, is defined for a composite event e as follows:

e :== e_1 Rel e_2

Rel :== temporal_relation | spatial_relation | sp_temp_distance | boolean_op

temporal_relation :== "after" | "before" | "simultaneously"

spatial_relation :== R_{i_j}

i, j :== [1,13]

boolean_op :== "AND" | "OR" | "NOT"

2.2 Spatiotemporal Compositions of actors

The "action" concept is related to the presentation of *actors* (multimedia objects) participating in IMDs. In the past, the term *synchronization* has been widely used to describe the temporal ordering of actors in a multimedia application [LG93]. However, a multimedia application specification should describe both temporal and spatial ordering of actors in the context of the application. The spatial ordering (i.e. absolute positioning and spatial relationships among actors) issues have not been adequately addressed. We claim that the term "*synchronization*" is poor for multimedia applications and, instead, we propose the term "*composition*" to represent both the temporal and the spatial ordering of actors.

Many existing models for temporal composition of multimedia objects in the framework of an IMD are based on Allen's relations [A83]. Nevertheless, these relations are not suitable for composition representation, since they are descriptive (they do not reflect causal dependency between intervals), they depend on interval duration and they may lead to temporal inconsistency. More specifically, the problems that arise when trying to use these relations are the following [DK95]:

- The relations are designed to express relationships between intervals of fixed duration. In the case of multimedia applications, it is required that a relationship holds independently of the duration of the related object (i.e. the relationship does not change when the duration changes)
- Their descriptive character does not convey the cause and the result in a relationship.

In the next subsections we refer briefly to a model for spatiotemporal compositions of actors [VTS98] that we have exploited for the definition of the IMD scenarios.

2.2.1 Temporal and Spatial Relationships and operators

In this section we refer briefly to temporal aspects of actor composition. We exploit the temporal composition scheme as defined in [VTS96]. We briefly introduce a similar scheme that also captures the causality of the temporal relationships. In this scheme the start and end points of a multimedia instance are used as events. Moreover, the well-known *pause* (temporarily stop execution) and *resume* procedures (start the execution from the point where the pause operator took place) are also taken into account.

Hereafter we present a set of TAC operators that represent the temporal composition of actors, together with the causality of temporal relationships among presentation intervals. For more details refer to [VTS96]. These operators correspond to the well known TAC actions: start (>), stop(!), pause (||), resume (|>), fast forward (>>) and rewind (<<). Therefore:

```
TAC_operator :== ">" | "!" | "||" | "|>" | ">>" | "<<"
```

(We have not defined an operator for the random positioning in time action as it would require an argument to denote the time point. We implement this action by defining an attribute for actors that specifies the point from which the actor should start playing).

We also have to illustrate the events arising from the temporal state changes of an actor, i.e. when object A starts its presentation then the "A>" temporal event is raised. Special attention should be paid to the event generated when the actor finishes its execution naturally when there is no more data to be presented ("<") and to distinguish this event from the TAC operator "!". Therefore:

```
t_event :== ">" | "<" | "|>" | "||" | ">>" | "<<"
```

We define now temporal composition representation. Let A, B be two actors, then the expression: A t_event t_interval TAC_operator B, represents all the temporal relationships between the two actors, where t_interval corresponds to the length of a vacant temporal interval. Therefore:

```
temporal_composition :==
                (Θ | object [{temp_rel object}]
temp_rel:== t_event t_interval TAC_operator
```

For instance: the expression: "Θ >0> A >4! B <0>C" conveys: ("zero seconds after the start of the application start A, 4 sec after the start of A stop B. 0 seconds after the end of B start C")

Finally, we define the duration d_A of a multimedia object A as the temporal interval between the temporal events A> and A<.

Another aspect of object composition in IMDs, is related to the spatial layout of the application, i.e. the spatial arrangement and relationships of the participating objects. The spatial composition aims at representing three aspects:

- The topological relationships between the objects (*disjoint*, *meet*, *overlap,* etc.)
- The directional relationships between the objects (*left*, *right*, *above*, *above-left,* etc.)
- The distance characteristics between the objects (*outside* 5cm, *inside* 2cm, etc.)

3 Spatiotemporal Composition Model

An IMD scenario presents media objects composed in spatial and temporal domains. A model that captures these requirements is presented hereafter.

For uniformity reasons, we exploit the Spatiotemporal origin of the image, Θ, that corresponds to the spatial and temporal start of the application (i.e. upper left corner of the application window and the temporal start of the application). Another assumption we make is that the objects that participate in the composition include their spatiotemporal presentation characteristics (i.e. size, temporal duration etc.). Hereafter, we define the spatiotemporal model:

Definition: Assuming two spatial objects A, B, we define the *generalized* spatial relationship between these objects as: sp_rel = (r_{ij}, v_i, v_j, x, y), where r_{ij} is the identifier of the topological-directional relationship between A and B, v_i, v_j are the closest vertices of A and B respectively (as defined in [VTS96]) and x, y are the horizontal and vertical distances between v_i, v_j.

We define now a generalized operator expression to cover the spatial and temporal relationships between objects in the context of a multimedia application. It is important to stress the fact that in some cases we do not need to model a relationship between two objects, but to represent the spatial and/or temporal position of an object relative to the application spatiotemporal origin, Θ (i.e. object A to appear at the spatial coordinates (110, 200) on the 10th second of the application).

Definition: We define a composite spatiotemporal operator that represents absolute spatial/temporal coordinates or spatiotemporal relationships between objects in the application: ST_R(sp_rel, temp_rel), where sp_rel the spatial relationship and temp_rel the temporal relationship as defined above.

The spatiotemporal composition of a multimedia application consists of several independent fundamental compositions. In other words, a scenario consists of a set of acts that are independent with each other. The term “independent” implies that actors participating in them are not related explicitly (either spatially or temporally). Though, there is always an implicit relationship through the origin Θ. Thus all compositions are explicitly related to Θ. We call these compositions *composition_tuples* and these include spatially and/or temporally related objects.

Definition: We define the composition_tuple in the context of a multimedia application as: composition_tuple :== Ai [{ ST_R Aj}], where Ai, Aj are objects participating in the application, and ST_R is a spatiotemporal relationship (as defined above).

Definition: We define the composition of multimedia objects in the context of multimedia applications as a set of composition_tuples: composition = Ci{,Cj}, where Ci, Cj are composition_tuples.

The EBNF definition of the spatiotemporal composition based on the above is as follows:

```
composition :== composition_tuple{[,composition_tuple]}
composition_tuple :==
                    (Θ | object) [{spatio_temporal_relationship object}]
spatio_temporal_relationship :==
                    "[("[sp_rel")","("temp_rel")]"
sp_rel :==  "(" rij "," vi "," vj "," x "," y ")"
x :== INTEGER
y :== INTEGER
temp_rel1:== t_event t_interval TAC_operator
```

where r_{ij} denotes a topological-directional relationship between two objects and vi, vj denotes the closest vertices of the two objects (see definition above). The term action is defined on previous sections.

3.1 The scenario model

The term scenario in the context of IMDs stands for the integrated behavioral contents of the IMD, i.e. what kind of events the IMD will consume and what actions will be triggered as a result. The scenario, in the current approach, consists of a set of autonomous functional units (*scenario tuples*) that include the triggering events (for starting and stopping the scenario tuple), the presentation actions to be carried out in the context for the scenario tuple, related synchronization events and possible constraints. Initial efforts for definition of scenario tuples may also be found in [VM93, VS96]. More specifically a scenario tuple has the following attributes:

[1] Specifically in the current implementation we adopted the “/\” operator. Then the composition A/\B that corresponds to the expression (A>0>B);(A<0!B);(B<0!A), may be expressed in natural language: “Start A and B and when the temporally shorter of the two stops the other”.

- Start_event: represents the event expression that triggers the execution of the actions described in the Action_list.
- Stop_event: represents the event expression that terminates the execution of this tuple (i.e. the execution of the actions described in the Action_List before its expected termination.
- Action_List: represents the list of synchronized media presentation actions that will take place when this scenario tuple becomes activated. The expressions included in this attribute are in terms of compositions as described in previous sections and in [VTS96].
- Synch_events: refers to the events generated (if any) at the beginning and at the end of the current tuple execution. These events may be used for synchronization purposes.

The scenario tuple is defined as follows:

```
scenario:== scenario_tuple [{,scenario_tuple}]
scenario_tuple :== Start_event ',' Stop_event ',' Action_List ',' Synch_events
Start_event :== Event
Stop_event :==  Event
Action_List :== composition
Synch_events :== '(' start, end ')'
start :== Event | " "
stop :== Event | " "
```

In APPENDIX A we present a sample IMD scenario with rich interaction and composition features. One of the parts of the scenario should adhere to the following verbal description:

> "*The next set of media presentations ("Stage 2b") is initiated when the sequence of events _IntroStop and _ACDSoundStop occurs. During Stage2b the video clip KAVALAR starts playback while the buttons NEXTBTN and EXITBTN are presented. The presentation actions are interrupted when any of the events _TIMEINST and _NextBtnClick occurs. The end of Stage2b raises the synchronization event _e1*."

The IMD scenario model may represent this functionality by the following scenario tuple definition:

```
TUPLE  Stage2B
    Start Event = SEQ(_IntroStop;_ACDSoundStop)
    Stop Event = ANYNEW(1;_TIMEINST;_NextBtnClick)
    Action List  = KAVALAR> 0 NEXTBTN> 0 EXITBTN>
    Synch Events = (_, e1)
```

In the following sections we use interchangeably the terms "composition tuple" and "instruction stream".

4 Implementation experiences

The IMD scenario model as presented in the previous sections gives an abstract view of the issue. We have implemented a set of tools for authoring and rendering IMDs. The authoring tool was implemented using Openscript (the scripting language of the authoring tool Toolbook) under Windows 95 and the rendering tool was implemented in Java. Although both tools support the vast majority of the modeling primitives some aspects were left out and some others were simplified. In the current status the following changes have been made:

- the spatial relationships between actors and events were left out
- the syntax of composition_tuples was simplified

We shall elaborate on the second change as the first one is quite straightforward. The old syntax distinguished between the notion of a TAC operator and an event. The operator was provoked by the user while the event was triggered because of the operator. We have adopted a new syntax that does not involve events. Therefore composition_tuples are now described as:

```
composition_tuple :==
            object [{temporal_relationship object}]
temporal_relationship :== t_interval TAC_operator
```

For example the action list of the tuple *Stage2b* mentioned in the previous example would be written as: "KAVALAR> 0 NEXTBTN> 0 EXITBTN>" (i.e. "KAVALAR start, 0 seconds after that NEXTBTN start, 0 seconds after that EXITBTN start").

4.1 Authoring

In this section we present the implemented authoring system based on the IMD model presented above. The authoring procedure is carried out in three consecutive phases. The IMD author may move in an iterative way between these three phases. The authoring phases are clearly distinguished and are the following:

- Selection and transformation of the media objects to participate; the author selects the media objects to participate in the IMD and defines the spatial and temporal transformations of the objects in the context of the current IMD.
- Definition of the events (atomic and complex) that the IMD session will consume; the author defines the atomic events that the current IMD will exploit for the scenario.
- Specification of the scenario of the IMD session in terms of scenario tuples. The scenario consists of autonomous units (scenario tuples as defined in previous sections), which consist of the event expression that activates the tuple when it occurs, and the expression that de-activates the tuple, if it occurs, prior to its expected end. The media objects to be presented in a synchronized way (in space and time) are included in the Actions List attribute of the scenario tuple.

The authoring process is an iterative loop between the aforementioned phases. The authors may refine/redefine their design and produce the desired IMD functionality. The relationships among the authoring phases are depicted in Figure 1. We will elaborate on the authoring procedure phases in the following sections.

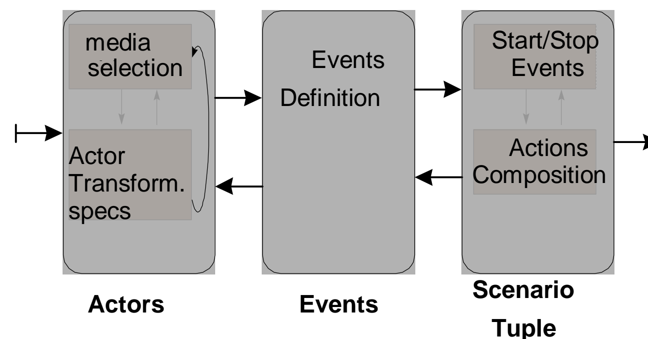

Figure 1: The authoring phases and their relationships

4.1.1 Actor specifications and transformations

As mentioned, *actors* are the entities representing the media objects participating in the IMD along with their spatiotemporal transformations [V96]. The objective of this phase is the definition of the actors, i.e. selection of the media objects to participate in the IMD as well as their *spatiotemporal features* and *transformations* in order to align to the authors requirements (see Figure 2). The author selects the media objects to participate in the IMD. This is carried our by selecting the “Add” button. Pressing the “Preview” button the author may preview (view an image, play a sound or a video) the object under concern. At this point the author is able to transform the object for the purposes of the current IMD.

Figure 2: Definition of an actor with both spatial and temporal features

These transformations are related to the object's spatial and temporal features (see Figure 3). The author may define a mnemonic name for the actor ("Actor_id"), the corresponding media file and the path in which it resides. With regards to the *spatial features*, the author may define the position in the application window ("Xcoord", and "Ycoord" coordinates of the upper left corner of the media objects relative to the upper left corner of the application window), and the size of the object ("Height" and "Width") in terms of screen units. As for the *temporal features* (which applies in the case of time-dependent media objects like sound and video) the author may define: the presentation direction (forward or reverse), the temporal portion of the object to participate in the application ("Start At" and "Duration" attributes) and the playback speed ("Scale" factor, essentially the temporal scaling). More refined control over the second feature may be achieved through the "Timing" button leading to the dialog box of Figure 4.

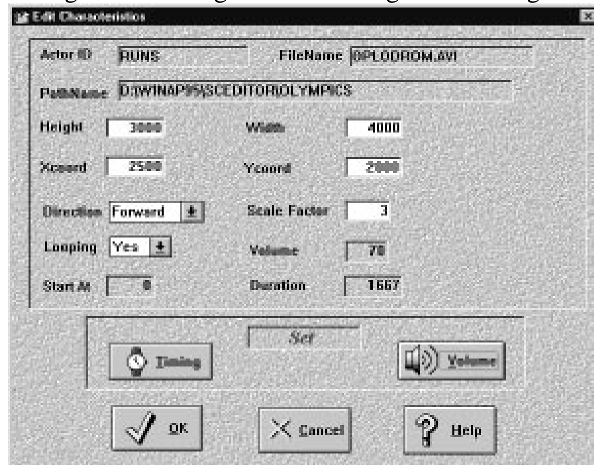

Figure 3: Definition of an actor with both spatial and temporal features

Figure 4: Temporal actor length definition

In the above dialog box the user may select the exact temporal portion of the media object under concern while previewing it. This step leads to the selection of actors and the proper definition of all their presentation features. The next step would be the definition of the events that our IMD will be interested in and will consume upon their occurrence. The result of the actor specification process is a declarative script. Such script appears in the sample application in APPENDIX A.

4.1.2 Event specification

The IMD may be considered as a context where events occur and are consumed according to the interest of the IMD. The authors may want to define happenings that are of interest to the IMD in order to trigger presentation actions. The authoring system provides a flexible event definition framework based on the model defined in [VB97] and presented in previous sections. The author may define a new event in terms of its *subject* (i.e. the actor that generates the event) and action (i.e. the action(s) of the actor that generates the event). The related dialog box is presented in Figure 5. There the event "_TIMEINST" is defined and raised when the actor TIMER1 counts 50 sec. At this stage only simple events may be defined.

Figure 5: Definition of the atomic events that will be consumed in the application

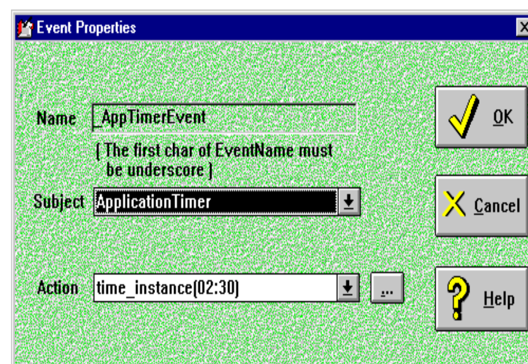

Figure 6: Editing the event properties.

In Figure 6 the event specification process is depicted. The user provides a name for the current event. The event is checked in the IMD context for uniqueness (as for its name) and, then, the author may select one of the actors (defined in the previous phase) as the subject of the event. Depending on the type of the actor (time-dependent, button, timer etc.) there is a different repertoire of actions available that may be selected at the “Action” dropdown list. Apart from the IMD specific media actors, there is a set of actors available to all IMDs. They are the following (the corresponding repertoire of actions appear in brackets): Application Timer (time instance, started at, Application start), System Settings (time, date), Keyboard (a set of keys like “Enter”, “Esc” etc.) and Mouse (“Left Button Click”, “Double Click” etc.). The user may select the “Action” that will generate the event under concern.

In the sequel, the scenario of the IMD may be defined. As scenario we perceive the behavioral contents of the IMD, i.e. what presentation actions will take place in the IMD context and how the IMD will respond to external (user, system) or internal (IMD actors) interaction. This is the objective of the third authoring phase described in the next section. In APPENDIX A the reader may find a sample set of events resulting from this authoring phase.

4.1.3 Scenario tuples’ specifications

The scenario tuple specification process that will be described hereafter is based on the model presented in [VM93, VS96] and also in previous sections.

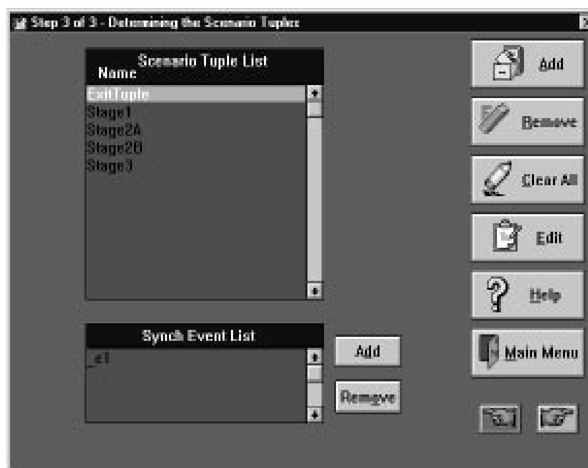

Figure 7: Definition of scenario tuples

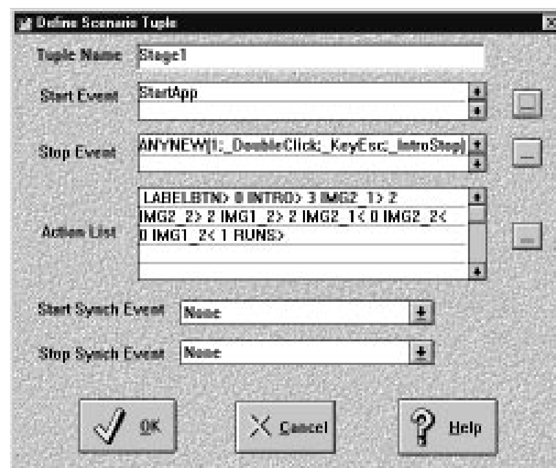

Figure 8: The attributes of a scenario tuple.

The user interface for scenario tuples manipulation appears in Figure 7. At this point, the list of already defined tuples and the sync_events (of the selected tuple) appear. The scenario definition tuple procedure appears in Figure 8. There the user may provide the name of the scenario tuple (which must be unique in the IMD context), then the “Start Event” and “Stop Event” may be defined.

The dialog box appearing in Figure 9 facilitates this purpose. At that point the author may define an arbitrarily complex event expression using the available composition functions and operators defined in earlier sections and presented in detail in [VB97]. The complex event consists of one or more expressions separated by brackets. The authring methodology forces syntactic correctness since the user may only select terms and adds them to the event expression rather than typing free text. During event definition, syntactic checking is done and errors are reported before leaving the Start/Stop event definition procedure. The user may edit each bracket separately.

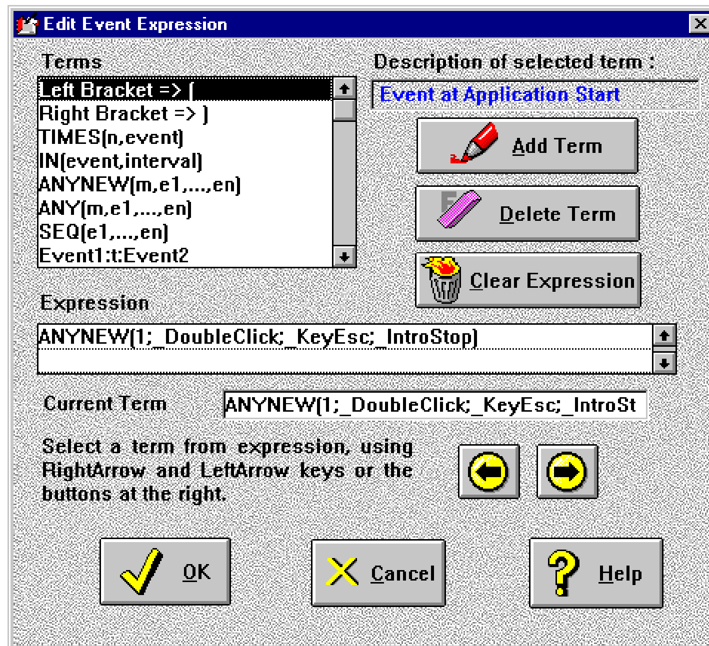

Figure 9: Editing the event expression for start and stop events

The next step is the definition of the actions that will be triggered by the activation of the scenario tuple. These actions are essentially an expression that includes synchronized presentation actions of the IMD actors. The temporal ordering and the relationships between the presentations actions are defined in terms of the operators set defined in previous sections. The author selects an actor from the "Actor list" (see Figure 10). Then, the available set of operators for the selected actor types appear in the "Operators" list and the author may select any of them. The author may as well add the temporal interval between two actor operators, whenever applicable. The action expression is again organized in independent expressions, enclosed in brackets, that are conjunctive and their contents are simultaneously triggered upon tuple activation.

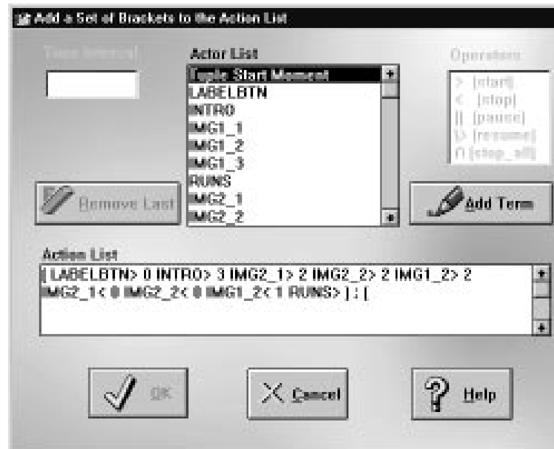

Figure 10: Adding action expressions in the "Action List"attribute of the scenario tuple.

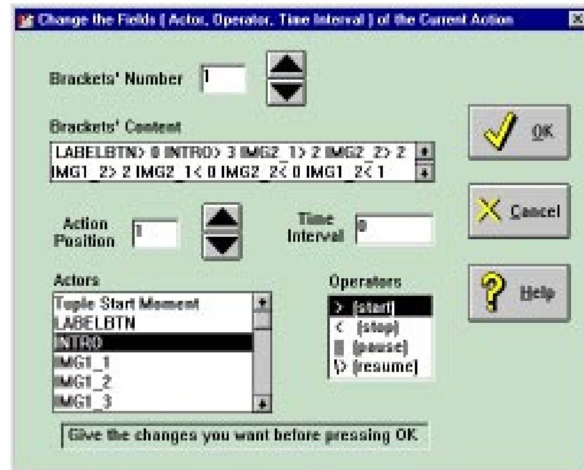

Figure 11: Editing an action expression in the action attribute of the scenario tuple.

It is important to stress the syntactic check that takes place before leaving this stage. Any errors are reported and the user is prompted to fix them.

The editing procedure of each bracket is carried out separately; the related user interface appears in Figure 11. At that point the user may select the actors to be presented (from the "Actors" list), the order of presentation ("Actor Position"), the temporal relationships among these presentations (from the "Operators" list) and the involved temporal interval length ("Time Interval").

The result of the above authoring process is a declarative script. Such script appears in the sample application in APPENDIX A. This script needs to be transformed into algorithms in order to be exploited for

building the IMD executable version. This task is accomplished by a a module that transforms the declarative script into algorithms so that the IMD may be executed.

An important issue in such a complex spatial and temporal composition is that inconsistencies may arise. We tackled this problem as regards pre-orchestrated scenarios in a fundamental way at object and IMD level, in [KSV99]. Though, when interactions are considered the consistency checking problems are much harder to tackle with due to the indeterminate nature of interactions. We addressed this problem in other research efforts [MPV99]. Currently, as regards integrity checking, mostly the temporal domain is addressed while the spatial one is an issue for further research

4.2 Rendering architecture

The IMD scenario model described so far is primarily related to the definition of a scenario and its components. In order to present (render) the media objects according to the scenario tuples (i.e. starting and stopping synchronized presentation of media objects), we must create a rendering scheme capable of presenting a scenario in such a way that all properties of the model are satisfied. It is essential to state the requirements of such a rendering scheme before continuing with its design. A rendering scheme should be able to do the following:

- detect events generated by the system, the user or the actors
- evaluate these events against the start/stop event expressions of each scenario tuple
- activate the appropriate tuples asynchronously
- execute all instruction streams concurrently
- perform synchronized presentation actions according to the scenario tuples' specifications.

The implementation framework should inherently support the following features:

- concurrent processes (i.e. a set of instruction streams running in parallel)
- dynamic object creation and manipulation in order to be able to support explicitly the theoretical model presented before as it is clearly defined over a set of separate entities (i.e. scenarios, tuples, actors, events)
- distribution features, as the objects may be located in any place (URL) on the Internet.

In that respect we have chosen to use Java, as this is a language that clearly satisfies the previous requirements. Java and other accompanying technologies offer many appealing features such as built- in multi-thread support, cross-platform compatibility. Moreover all WWW browsers support Java to a great extent making, thus, the presentation of an IMD feasible in any WWW browser. At this point we are ready to present the architecture of the rendering scheme.

We implemented the system in a client-server approach. Both the client and the server, are implemented in Java and are, therefore, portable through a variety of platforms. The communication between them is performed exclusively using the RMI (Remote Method Invocation) protocol [J97a], a technology to seamlessly distribute Java objects (in our case IMDs and media objects) across the Internet and Intranets. RMI allows Java objects to call methods of other Java objects. These methods can return primitive types and object references, but they can also return objects by-value, a feature unique to RMI among its related technologies (CORBA, DCOM). The continuous media, video and sound, are retrieved by the client from http servers specified in the scenario, and presented with the aid of Java Media Framework (JMF) [J97b] which specifies a unified architecture, messaging protocol and programming interface for media rendering, media capture, and conferencing. JMF APIs support the synchronization, control, processing, and presentation of compressed streaming and stored time-based media, including video and audio. Streaming media is supported, where videos and sound are reproduced while they are being downloaded without being stored locally. This feature is extensively exploited in the system described here. The IMD script (actors, events, scenario tuples) produced by the authoring phase, is parsed and translated into corresponding Java objects that can be stored in the server using the Java Serialization services [J97a]. Finally the client may request a scenario from the server, retrieve and present it to the user by retrieving the media from the appropriate http servers.

4.2.1 The Server

The server's task is to provide the client with the scenarios it requests, as well as all the media used by the scenarios. In order to minimize download latency and space requirements at the client side, continuous media are requested by the client only at the moment they are to be presented. This depends on user interactions and the scenario itself. As soon as they are asked for, the media (audio, video) are sent to the client which presents them on the fly, without storing them locally. The server system is composed of two separate entities (Figure 12):

The distribution of scenarios is handled by the Multimedia Document Server (MDS). It is responsible for publishing the directory of scenarios it can serve, as well as serving the scenarios themselves. At this level, all communication with the client is performed by remote method calls, through the Java RMI registry. All scenarios all implemented as serializable Java objects, stored in a file at the server side. The client requests a specific scenario through a method call, and receives it as the return value of the method. This avoids the need to create a dedicated application level protocol and simplifies the implementation, as well as the updating of the client and the server. The need to pass objects by-value was the primary reason for selecting RMI among related technologies (CORBA, DCOM), the next being its high integration with the Java environment.

The distribution of media is handled by an array of http servers: as soon as the client has downloaded the scenario (which includes presentation and interaction information, and only *links*, in the form of URLs, to the media it presents), it parses and executes it. When an image, video, sound or text is called for by the scenario, the client communicates directly with the http server that stores it and retrieves it. In the case of static media (text, images), they are downloaded and presented according to scenario specifications. However, when continuous media (video, audio) are to be shown, they are played directly from their site of origin and presented on the fly. To make this possible, the Java Media Framework is used. The decisive factor on the selection of JMF over other similar technologies was that it needs to be present only at the client side—on the server side, any http server can be used, without any special configuration. This permits the massive distribution and replication of media to any number of http servers, thereby reducing load on any individual server, whereas the scenarios (time-independent and usually only a few kilobytes in size) can all be served by one machine.
An issue with JMF is that it does not currently provide QoS guarantees, neither at the http servers, nor for the transfer of data. The latter could be overcome using a resource reservation protocol such as RSVP [RFC2205] in the network path between the client and the server, while the former is dealt with, to a limited extend, by distributing the media to a number of http servers.

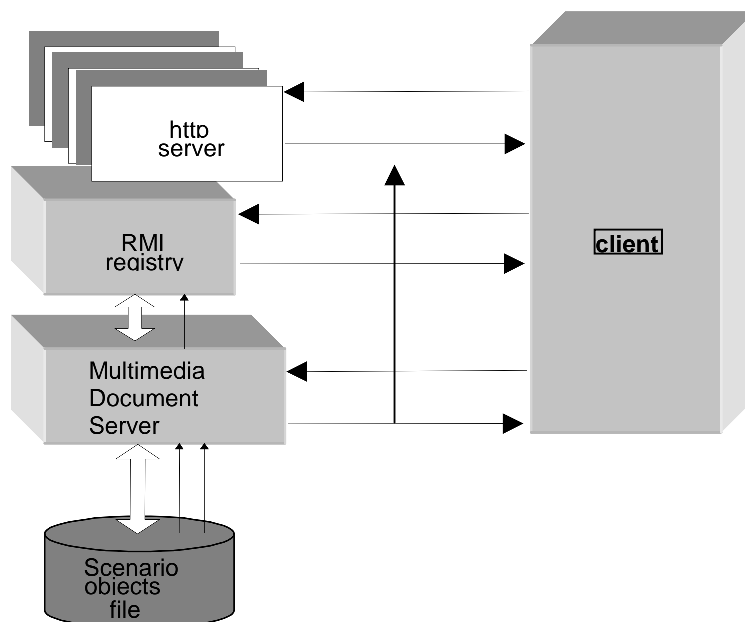

Figure 12. The server architecture and the communication with the client

Another interesting option offered by this schema, is the ability to use http servers owned and administered outside the scenario creation team: an interactive presentation of a country could, for example, include a video on the latest events in this country, by means of a link to the respective video at the web site of, say, CNN. The video would be updated daily without the need to make any changes to the scenario, or to maintain a privately owned web server.

4.2.2 Rendering scheme architecture - the Client

Our scheme mainly consists of two sets of modules. The first includes the modules aiming at presenting a scenario, while the second one is composed of the modules in charge of event detection and evaluation. These two sets work together in order to present a scenario to the end user. The basic idea behind this design is to use the first set to start a scenario, and then use the second one to control the flow of presentation for that scenario. We shall now elaborate on the architecture describing the implemented Java classes of our

modules. Each element of a scenario (i.e. tuples, events and actors) is represented as a different Java class. Therefore, we have a set of classes as follows:

- *ScenarioPlayer* class and a *TuplePlayer* class: are capable of handling an IMD scenario and a scenario tuple respectively
- *Actor* class: serves as the super-class of the classes that correspond to presentable media objects and user interface elements, namely: *Video*, *Image*, *Sound*, *Text*, *Button*, *Label* and *Timer*
- *AppEvent* class: stores information on all the events that may occur in the specific IMD session.

Apart from these we have also defined a few other classes that are essential to our scheme. The *InstructionStreamPlayer* class is responsible for synchronized presentation of media objects. We also have a set of listener classes that are in charge of presentation of a single media object and detect all events related to the single object presented. As an example, in order to show a video on the screen we should use an instance of the *VideoListener* class. Furthermore, that particular instance would have to be responsible for detecting all events associated with the video, such as the end of the video. Another fundamental class in our design is the *EventEvaluator* class, which is in charge of evaluating the start and stop events of all involved tuples each time a simple event occurs, and sending the appropriate messages to the *ScenarioPlayer.*

When an IMD scenario is to be presented, the following steps are executed. First, a *ScenarioPlayer* and an *EventEvaluator* object are created. The former generates a "StartApp" (start application) event. This is a special kind of event that denotes the start the IMD session and may appear in one or more tuples as their start event, meaning that the action part of the tuples will start playing as soon as the scenario starts. This event is sent to the *EventEvaluator* that determines which tuples are to be started and tells the *ScenarioPlayer* to start them. The *ScenarioPlayer* then creates the corresponding *TuplePlayer* objects. Each *TuplePlayer* creates as many *InstructionStreamPlayer* objects as necessary. The *InstructionStreamPlayer*s present media objects according to the scenario specifications by creating the appropriate listener objects. A listener object is created for each media object presented and each listener is paired with an instance of the *EvaluatorNotifier* class. This class's sole purpose is to furnish the listeners with a set of methods to send messages to the *EventEvaluator.* For example, when a *VideoListener* or a *SoundListener* detects the end of their corresponding media object they use the "sendStopEvent" method of the *EventEvaluator* class to pass that message to the *EventEvaluator*. As the IMD session continues, more actors start and stop and events keep generating. *EventEvaluator* collects all these events and in collaboration with *ScenarioPlayer* defines the flow of the presentation. This is achieved by having the *ScenarioPlayer* start or stop the appropriate tuples, by creating new *TuplePlayer* objects or by destroying the ones that should stop. When a tuple must be interrupted, all the participating actors are interrupted (if they are active) and the appropriate synch event (if one exists for the tuple) is sent to the *EventEvaluator*. Finally, there comes a point that a tuple with action "ExitApplication" --another special action that denotes the end of a scenario-- starts and automatically all media stop playing and the scenario finishes. The procedure so far and the interdependencies among the instantiated classes are depicted in Figure 13.

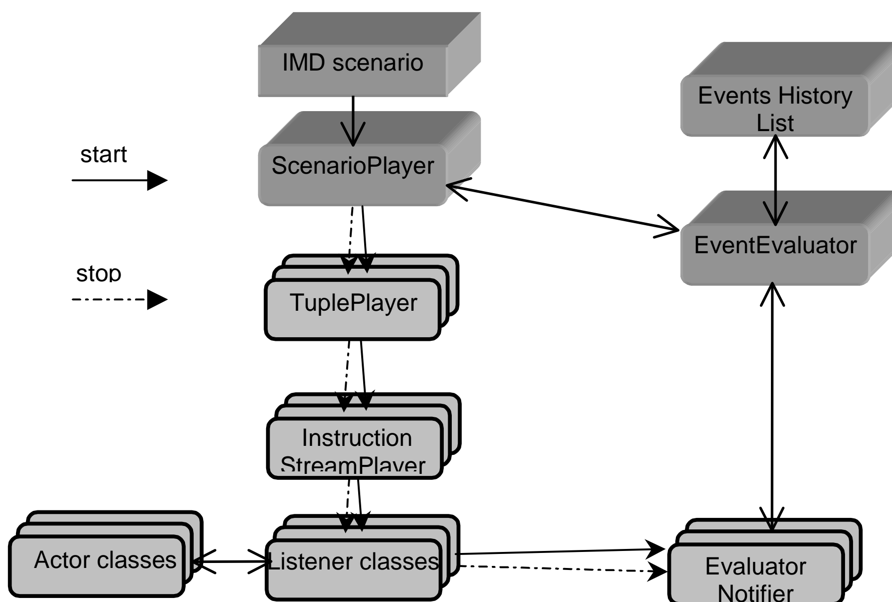

Figure 13. The architecture of the client for IMD scenario execution

In this figure, we also notice the existence of a class called *HistoryList*. This class is like a storage facility that holds information about all events occurred so far. The events stored in this class are either simple events or synchronization events as defined above.

One of our main concerns when designing this rendering scheme, was to be able to detect all events generated by the IMD sessions. We would also like to minimize the effect of an object failure in the IMD session (i.e. the site holding the media data goes down) in order to avoid side effects to other actors so that scenario presentation would not crash. Therefore, we have implemented the *TuplePlayer, InstructionStreamPlayer,* listeners, *EvaluatorNotifier* and *EventEvaluator* classes as threads. The first three guarantees us that all actors are separate entities that do not affect one another; the two remaining classes make certain that we detect all events.

The rendering scheme includes two main tasks: detecting and evaluating all events, so as to start and interrupt scenario tuples on the basis of these events, and presenting media objects according to the specifications of the instruction streams in the scenario tuples. In the following subsections we present these tasks in detail.

4.2.3 Event handling

Detecting events

As mentioned earlier in the IMD model, we deal with four categories of events: inter-object, intra-object, user and application events. At this point our rendering scheme is able to detect all kinds of events except intra-object events. The reason for this are the constraints imposed by the existing Java language facilities; though, this may change in the future. Hereafter we shall examine the way our system handles each event category in detail.

Inter-object events

This is the most significant category of events as it encompasses all events corresponding to media objects, such as sound, image and video. We have already stated that for each media type, there is a different listener class capable of detecting intra-object events related to the object. For example, a listener starts a video object, namely an instance of the *VideoListener* class that is able to detect that event. The listener will handle that event and decide whether or not the event is significant for the scenario. If the event is of interest to the scenario (i.e. it is included in one of the start/stop event expressions), then the listener will dispatch that event to the *EventEvaluator* with the use of the *EvaluatorNotifier*. In this stage the listener not only detects all events concerning a media object but also performs some kind of filtering by discarding all events irrelevant to the scenario. This is clarified in the following examples:

Assume a video named “v1”. When the video starts execution (in section 3.2.5 we deal with this issue in more detail) then it will send a message to its’ paired *VideoListener*. The listener will collect that event and ask the *ScenarioPlayer* if there is an event with subject = “v1” and action = “start”. If such an event exists (i.e. it is an interesting event for the IMD), then the listener will send the event together with the actor that caused it (“v1”) to the *EvaluatorNotifier*. The latter will in turn update the internal storage structures of our scheme (to be examined later) with the new state of video “v1” and then forward the event to the *EventEvaluator* for processing.

By using a different listener class for each media type, we guarantee maximum flexibility in defining detectable events and the classes are more viable to modifications due to Java as we only have to correct a small piece of code. By having a separate instance of the appropriate class for each media type, we minimize the consequences of a media-object problem that may occur (for instance a network problem).

User-events

User events are either mouse events or keyboard events. As these events are not related to a specific media type, we adopt a different strategy for their evaluation. The key point with this kind of events is that there is no indication of when they will happen and that they cannot be assigned to any actor. For example a user event may be generated when the user presses key “F2” and this may occur many times during and IMD session. Thus, we must find a new way of capturing these events. The approach we adopted was to create a number of user-event listener objects right from the start of the IMD sessions. For example, if the scenario author has defined three user-events: the pressing of key “F2”, a mouse click and a mouse double-click, then we shall create three listeners: one for “F2”, one for the click and one for the double-click. These listeners are implemented as a part of the Java language. Their instantiation occurs right after we create the instance of the *ScenarioPlayer* class.

Once a user-event occurs, the action course is somewhat different to the one in the case of intra-object events. The event is not sent to an *EvaluatorNotifier*, as there is no need to check whether or not it is of interest to the scenario. We are certain it is of interest as we have only created listeners for events mentioned in the scenario events list (see section 3.2.4). Thus, these events are forwarded directly to the *EventEvaluator* for processing.

Application-events

In this case, we are interested in detecting events such as "the system time is 14:00", or events generated by timers. A timer is an actor type that expires at a certain point in time. This point is set when creating the timer (i.e. we can create a timer that will go off after 50 seconds). Timer events are dealt with in the same way as intra-object events. That is, each timer is paired with a *TimerListener* that waits for the timer event to happen and sends it to the *EventEvaluator* when it occurs.

Before concluding our discussion on event detection, we would like to remark that each event sent to the *EventEvaluator* is accompanied by a timestamp, indicating the exact point in time the event occurred. These timestamps are acquired by a general application timer that starts together with the scenario start relative to the IMD session start. The application timers' granularity is 1 second and indicates how many seconds have passed from the beginning of the scenario presentation.

Evaluating events

The next step is to evaluate the events occurred so far. All detected events are simple events and they are all sent to the *EventEvaluator*. This means that on arrival of a new event, the start and stop event expressions (complex events) of all tuples should be evaluated. Those that are found to be true trigger the appropriate actions. This task may be complex since the start/stop event expressions may be arbitrarily complex, so an incremental evaluation as new events occur is necessary. The *EventEvaluator* additionally controls the synchronization of all modules that send messages to it about events that occurred. This function is further explained in the *EvaluatorNotifier* class presented in section 3.2.5.

Hereafter, we present the *HistoryList* class. There is only one instance of this class in each IMD session and it keeps information on the events that have occurred in an IMD session, from the start to the current time, which is defined as the time elapsed since the session start. For each event we keep all the timestamps of its occurrences. For example, the entry: <e1, 2, 6, 12> implies that event "e1" occurred 3 times with the timestamps: 2, 6 and 12. The timestamps refer to the time elapsed from the IMD session start in seconds. It is important to clarify that in the *HistoryList* only simple and tuple synchronization events that have occurred in the current IMD session, are stored. In this structure there is no information on events that are included in the *Events* list (see section 3.2.4), but have not occurred up to the current moment.

Upon arrival of an event, the *EventEvaluator* locks itself (i.e. does not accept any further messages) until the event evaluation process finishes. During this process, the aim is to find all tuples that have to start or stop due to the occurrence of the current event. Once all tuples that must either start or stop have been found, the *EventEvaluator* sends the appropriate messages to the *ScenarioPlayer* to act accordingly and continues processing any synch events that these tuples may have defined. In some cases an occurring event may cause both the start and the stop event of a tuple to be evaluated as true. If this is the case, then the system allows a tuple either to start or stop, but not both. The choice is made according to the current state of the tuple.

On evaluating an event expression as TRUE, several actions may be taken such as starting/interrupting a tuple or interrupting the IMD session. The evaluation process is carried out by a set of functions that are further presented. First, we are interested in the evaluation of a simple occurring event. This task is accomplished by the function evaluate_simple_event() invoked each time the *EventEvaluator* receives an event (this may be a simple event or a synchronization event). The algorithm follows:

```
EventEvaluator receives an event e
EventEvaluator “locks” itself
evaluate_simple_event(event e) {
EventEvaluator writes e to HistoryList
for each tuple t
        if t is idle
        then {
                evaluate (t.start_event, e)
                if t.start_event is true
                        then add t in tuples_to_start array
                }
        else
        if t is active
        then {
                evaluate (t.stop_event, e)
                if t.stop_event is true
                        then add t in tuples_to_stop array
            }
end for
start_tuples(tuples_to_start)
stop_tuples(tuples_to_stop)
}
EventEvaluator “unlocks” access to others
```

It is important to stress that during the period of event processing (*EventEvaluator* in locked state) the occurring events are not lost but are maintained in the respective *EvaluatorNotifier*s. When the *EventEvaluator* finishes processing an event, it notifies all *EvaluatorNotifiers* that may have a new event to send, that it is now available to process a new event. It then receives a new event, falls again into the locked state and processes it. By having the *EventEvaluator* notify the *EvaluatorNotifiers* instead of having them poll the *EventEvaluator* at regular time intervals we have a significant gain in performance as the *EventEvaluator* is used as soon as it is needed.

The function evaluate(t.start_event, e) carries out the evaluation of the event e against the event expression stored in the start/stop event of the tuple. The resulting value will determine whether the tuple t should start/stop. The function will be presented in more details further in this section.

In the sequel, we will explain the way the evaluate() method works. When the start/stop event expression is a simple event, the evaluation is limited in searching in the HistoryList for an occurrence of such an event. For example, if a tuple must start when event “e2” happens, all we have to do when checking the start event expression of that tuple is to see if “e2” exists in the HistoryList[2].

In the case that the start/stop event expression is complex, it may contain expressions including operators and functions that are defined in the framework presented in [VB97]. Our current rendering scheme implements the AND, OR and NOT operators and the functions: ANY, ANYNEW, IN, TIMES, SEQ and (event1:time interval: event2). In this case the evaluation process is carried out in three distinct steps. The first step is the transformation of the event expression into postfix form resulting in an expression without brackets. The second step is evaluating each function appearing in the expression, and replacing the function with the token “true” or “false” if this is the case. The last step is to evaluate the result of the tokens “true” or “false” combined with the Boolean operators. The above-described steps are demonstrated in the following example.

Assume that the current time is 13secs after the start of the an IMD session, we are evaluating event “e5” and we are now processing the start event expression: “(e1 AND ANY (2; e1; e2; e5)) OR e4”. The contents of the *HistoryList* in the current session appear in Table 1.

Event	Timestamps
e1	2, 7, 12
e3	5
e5	13

Table 1: History list contents at time 13 sec.

Event	Timestamps
e2	3, 9
e3	7
e1	13

Table 2: The contents of the History list at time 13 sec in another session.

[2] This is not entirely true, as we first have to check if the event we are currently evaluating against the tuple start/stop event is “e2”, or some other irrelevant event (see the end of this section).

The *first step* will result in the transformation of the event into the expression (the symbol "/" serves as a delimiter): e1/ANY(2;e1;e2;e5)/AND/e4/OR

The *second step* will produce the expression: "true/true/AND/false/OR"

The *third step* will evaluate this expression to "true". If the tuple with the above event expression is idle (i.e. has not already started or has already finished), then it must be started.

A disadvantage with the evaluate_simple_event() method is that it evaluates the start/stop events of all tuples against each occurring event. This would lead to tuples that had previously been activated or stopped being re-started/re-stopped just because their start/stop event expression would evaluate true (the contents of the history list still include the events that had triggered the execution of these tuples in the first place).

We have dealt with this problem and we propose a mechanism that solves it. Assume tuple t1, and t1.start_event = "e2 AND e3" and that event e1 occurs at time 13sec with the HistoryList being as appears in Table 2. During the evaluation of e1, the expression "e2 AND e3" would evaluate to true, and provided that tuple t1 had finished playing, we would have to start it again. It is clear that event e1, which is under evaluation, is irrelevant to tuple t1 and it would make no sense to start tuple t1 because of an irrelevant event.

Therefore, we have expanded the evaluation mechanism to check whether the simple event (for instance e1) we are currently processing is related to the tuple whose start or stop event expression we are evaluating. If e1 participates in the start or stop event expression, then the evaluation process goes on. Otherwise, the evaluation stops. For instance, in the above example, e1 does not participate in the expression "e2 AND e3"; thus, the expression will not be further evaluated. This mechanism enables a tuple to be executed several times as its' start event expression will become true only when the proper event occurs.

4.2.4 Starting and interrupting scenario tuples

In order to accomplish this task, the client must detect and evaluate the events that occur in an IMD session and match them against the events included in the start/stop event attributes of the scenario tuples. A tuple is considered *active* when the start event of that tuple is evaluated as *true*. At this point all instruction streams of the tuple start execution at the same time, although they do not have to stop concurrently.

When an IMD session starts, none of its tuples is active. A special event called "StartApp" (start application) is generated and the tuples whose start event is the "StartApp" event, start their execution. A tuple cannot be restarted when it is active, even if its' start event becomes true. A tuple can only start again once it has stopped/finished and, thus, is in idle state.

Once a tuple has been initiated, there are two ways it may end: *natural* or *forced.* In the first case, the tuple falls into the idle state when all instruction streams have finished. An instruction stream state is considered as finished when all the involved actors have fallen into stopped state. In the second case, the tuple stops when its' stop event becomes true.

In order to avoid confusion, we explain what are the semantics of interrupting an actor. For this purpose we distinguish actors with inherent temporal features (sound or video) and actors without such features. An actor of the first category falls in the idle state either when its' natural end comes (there is no more data to be presented) or when it is stopped using the stop operator "!". Actors of the second category (e.g. an image) stop only when we apply the stop operator on them.

Hereafter, we will examine the classes that are related to management of scenario tuples, namely the *ScenarioPlayer* and the *TuplePlayer.*

The *ScenarioPlayer* class is responsible for the execution of an IMD. Initially, it constructs the window where the media are to be presented (a "frame" in Java jargon) and becomes able to receive all input events (keyboard or mouse generated) as well as all application timer events (the entity that counts the overall application time). This class is also responsible for starting and interrupting tuples. In order to achieve these tasks the *ScenarioPlayer* maintains information about the IMD session in structures supported by a set of auxiliary classes. The most important ones are:

- *Actors*: stores information on the participating objects and their spatiotemporal transformations in the current IMD.
- *Tuples*: stores the tuples of the IMD

- *Events*: stores information about the events that the IMD will consume. It includes only the simple events, while the potentially complex events that are defined as scenario tuple start/stop or synchronization events are stored in the corresponding tuples.
- *ActorsEnabled*: stores information on the actors that are currently active together with the identifier of the tuple that activated them and their activation time, referring to the temporal start of the IMD.
- *TuplesEnabled*: stores a list with the tuples that have been activated and/or stopped during the IMD session along with the activation/interrupting time points. It is clear that a scenario tuple may be executed more than once during a session, depending on the occurring events.

The *TuplePlayer* class is in charge of starting and interrupting a scenario tuple. In other words, it starts the instruction streams of the scenario tuple with no further effect on them. Each time a scenario tuple is to be started, the ScenarioPlayer creates a TuplePlayer object. When it is stopped/finished, the *TuplePlayer* must destroy the related occurrences of the *InstructionStreamPlayer* class. The *TuplePlayer* must detect the termination of the instruction streams that are included. When all instruction streams have finished, the *TuplePlayer* informs the *ScenarioPlayer.* At this point it is stopped.

4.2.5 Synchronized presentation of media-objects

In this section we present the classes of the client that are in charge of presenting the media objects according to the synchronization relationships that are included in the instruction streams. As mentioned above, each scenario tuple consists of a set of instruction streams. Not all instruction streams have the same effect on actor states. In this respect we distinguish two categories of instruction streams.

The first one includes instruction streams whose synchronization expression starts with an actor followed by start operator (>). These instruction streams start immediately after the tuple activation and remain active until all participating actors stop. The second category includes instruction streams that contain the synchronization operator "/\". These instruction streams also start immediately but remain active until the temporally shorter of the involved actors ends its execution. If an instruction stream contains the synchronization operator "/\", it cannot contain any other operator (i.e. >, !, ||, \>).

The *InstructionStreamPlayer* class is designed to execute an instruction stream as defined in earlier sections. The way this is done is by parsing the instruction stream string at execution time and executing the actions it says as we parse it. For example, assume the instruction stream: "video1> 4 image1> 0 button1> 5 video1||". It implies that the video clip "video1" should start, and after 4 seconds the image "image1" be presented. Immediately after the button "button1" is presented and after 5 seconds video1 is suspended.

The *InstructionStreamPlayer* starts parsing the instruction stream and finds string "video1". This string must be an actor (actually the ScenarioPlayer verifies the name, since this class maintains information about the actors). Once *InstructionStreamPlayer* gets the actor "video1", it continues parsing and finds that the start operator (">") is to be applied to video1. This is accomplished by creating a new *VideoListener* object, which starts the video presentation according to the specifications of the corresponding actor. While the *VideoListener* performs this task, the *InstructionStreamPlayer* continues parsing, finding the 4 seconds pause. This is accomplished by falling in the idle state (i.e. release the CPU) for 4 seconds, and then continue parsing. The same steps (find actor, find operator, apply operator to actor, and wait for a number of seconds) are repeated until the whole instruction stream is processed. A new listener object is created only when the start operator is found. For the other operators, the *InstructionStreamPlayer* does not create a new listener; instead it sends the appropriate message (pause, resume or stop) to the corresponding listener previously created.

In the case of synchronization expressions including the operator "/\" (for example assume "video1/\ button1/\ sound2/\ text3"), things are slightly different. All actors participating in the instruction stream are first inserted in an array and then the appropriate listeners are created for these actors. Each listener presents one media object and when the first objects finishes, the corresponding listener notifies the *InstructionStreamPlayer* which in turn sends messages to the other listeners to stop.

In order to present single media objects, we use a set of listener classes. A set of six classes (each for a different kind of actor) were created and all have the suffix "Listener". These classes do not only present actors but also detect ("listen to") any events concerning the actor they are controlling (i.e. media state changes etc.). For instance, the *VideoListener* class can start, stop, pause and resume a video clip, and can also detect all kinds of events that are related to the particular video. A *VideoListener* must receive appropriate messages to start, pause, resume and stop the video it is currently playing. The class is also in charge of presenting the video according to the specifications in the corresponding actor (i.e. volume

intensity, start point, screen coordinates etc). The same applies to the other listeners, namely *SoundListener*, *ImageListener*, *TextListener*, *ButtonListener* and *LabelListener*. Listeners, as mentioned before, also serve to pass the occuring events that are of interest to the IMD to the *EventEvaluator*.

5 Related Work

In this section we review related work that addresses modeling of IMDs and compare to our rendering approach.

5.1 Multimedia Document models, standards and rendering approaches

In this subsection we review Multimedia synchronization modeling approaches and Multimedia document standards regarding the interaction support they provide. In [IDG94] a model for spatiotemporal multimedia presentations is presented. The temporal composition is handled in terms of Allen relationships, whereas spatial aspects are treated in terms of a set of operators for binary and unary operations. The model lacks the following features: there is no indication of the temporal causal relationships (i.e., what are the semantics of the temporal relationships between the intervals corresponding to multimedia objects); the spatial synchronization essentially addresses only two topological relationships: overlap and meet, giving no representation means of the directional relationships between the objects (i.e. Object A is to the right of object B) and the distance information (i.e. object A is 10 cm away from object B); the modeling formalism is oriented towards execution and rendering of the application, rather than for authoring. In [H96] a synchronization model is presented. This model covers many aspects of multimedia synchronization such as incomplete timing, hierarchical synchronization, complex graph type of presentation structure with optional paths, presentation time prediction and event based synchronization. As regards the events, they are considered merely as presentations constrained by unpredictable temporal intervals. There is no notion of the events semantics and also no notion of composition scheme.

In [SKD96] a presentation synchronization model is presented. Important concepts introduced and manipulated by the model are the objects states (Idle, ready, In-process, finished, complete). Although events are not explicitly presented, user interactions are treated. There are two categories of interaction envisaged: buttons and user skips (forward, backward).

As mentioned in [BS96], event based representation of a multimedia scenario is one of the four categories for modeling a multimedia presentation. There it is mentioned that events are modeled in HyTime and HyperODA. Events in HyTime are defined as presentations of media objects along with its playout specifications and its FCS coordinates. As regards HyperODA, events are instantaneous happenings mainly corresponding to start and end of media objects or timers. All these approaches suffer from poor semantics conveyed by the events and moreover they do not provide any scheme for composition and consumption architectures.

MHEG [ISO93] allows for user interaction between the selection of one or more choices out of some user-defined alternatives. The actual selection by a user determines how a presentation continues. Another interaction type is the modification interaction, which is used to process user input. For the modification interaction a content object is offered to a user, e.g. to enter a text with which a video is selected or a number to change the volume of an audio. Temporal access control actions, however, cannot be modeled with MHEG. The hypermedia document standard HyTime [ISO92] offers no modeling of interaction. Object Composition Petri Nets (OCPN) [LG93] do not inherently support modeling of interaction.

HyTime [B96] provides a rich specification spatiotemporal scheme with the FCS (Finite Coordinate Space) space. FCS provides an abstract coordinate scheme of arbitrary dimensions and semantics where the author may locate the objects of an IMD and define their relationships. Nevertheless, the formalism is awkward and, as the practice has proved, it is rarely used for real world applications. Eventually Hytime is not an efficient solution for IMD development since “there are significant representational limitations with regard to interactive behavior, support for scripting language integration, and presentation aspects” [B96].

All the aforementioned models and approaches do not capture the rich semantics of events that occur in an IMD. Moreover, none of those approaches propose a composition scheme.

As regards rendering, related work has been published in [KE96]. Although it copes with multimedia objects, it models a smaller part of the IMDs that relate to the multimedia documents and not to multimedia applications. Thus, taking for granted that the document will be based on textual resources, the model tries to make an interactive multimedia “book” containing some form of multimedia objects like images, sound and video. The book is divided in chapters and the screen layout is similar to the one of word processors,

along with their temporal information. The temporal relationships are taken into account, but not the spatial ones since it is assumed that they are solved depending on the text flow on the page.
Another system is presented in [CPS98] for specification and rendering of Multimedia Documents. The approach presented there is based on presentation and retrieval schedules. In this paper there is no indication of interaction handling.

In [HCD98] an event based synchronization mechanism is presented based on the tool PREMO. This approach claims that it is appropriate both for event-based as well as for time-based presentations. The event handlers that are responsible for propagation of events handle only simple events.

Special attention should be payed to the upcoming standard SMIL. The key to HTML success was that attractive hypertext content could be created without requiring a sophisticated authoring tool. Synchronized Multimedia Integration Language (SMIL)[SMI98] aims at the same objective for synchronized hypermedia. It is an upcoming standard for synchronized multimedia to be presented in a WWW browser. SMIL allows integration of a set of independent multimedia objects into a synchronized multimedia presentation. A typical SMIL presentation has the following features:

- the presentation is composed of several components that are accessible via a URL, e.g. files stored on a Web server.
- the components are of different media types, such as audio, video, image or text.
- interaction support in terms of simple events. This implies that the begin and end times of different components have to be synchronized with events produced by internal objects or by external user interaction. Also simple user interaction is supported. The user can control the presentation by using control buttons known from video-recorders, such as stop, fast-forward and rewind. Additional functions are "random access", i.e. the presentation can be started anywhere, and "slow motion", i.e. the presentation is played slower than at its original speed.
- support for hyperlinks, the user can follow hyper-links embedded in the presentation
- rich high level temporal composition features such as lip-synchronization, expressive temporal relations (including parallel with a master, interruptions with the first ending element (par-min), wall-clocks, unknown durations, etc...

In [Bul98] the GRiNS authoring and presentation environment is presented. GRINS can be used to create SMIL-compliant documents and to present them. The temporal and spatial structure model supported is identical to the one of SMIL while a stand alone rendering module (called playout engine) can present the document.

SMIL in its current form does not support relative spatial positioning and spatial embedding of media objects, an aspect extensively covered by our authoring approach. Moreover, the temporal synchronization model does not address the causality of temporal relationships, while the interaction model is rather limited as regards the multitude of events that may occur in a presentation. The temporal composition is represented as a subset of the Allen relations (essentially the authors deal with sequential and co-start /co-end and equality in time relationships). Then a temporal graph is constructed and the global set of constraints is solved offering alternative durations for the objects and the whole presentation.

This approach lacks some features that are covered in others, such as: spatial specifications and interaction.
It rather verifies temporal features of the presentation than executing it.

In [Yu97] a Hypermedia Presentation and Authoring System suitable for the WWW is presented. In that effort the temporal synchronization is focused on definition of serial (i.e. objects one after the other)and parallel relationships (i.e. objects starting or stopping at the same time). Each object is considered to be active when it is presented on the screen and deactivated when it is removed from the screen. As for spatial synchronization, this approach defines the positioning of an object in a rectangular area that may be scaled for the needs of the presentation and special constraints are applied so that objects will not overlap during their presentation.

The approach does not handle the intermediate pause and resume actions, which are common in a presentation of time-dependent objects, while it does not handle the causality features of the temporal relationships (i.e. the end of video A causes the start of video B). Finally there is no indication of capabilities to represent spatial relationships and embedding whereas interaction handling is limited to anchors.

5.2 Active databases – event detection & evaluation

The concept of events is defined in several research areas. In the area of Active Databases [G94, GJS92, CM93] an event is defined as an instantaneous *happening of interest* [GJS92]. An event is caused by some action that happens at a specific point in time and may be atomic or composite. In the multimedia literature

events are not uniformly defined. In [SW95] events are defined as a temporal composition of objects, thus they have a temporal duration. In other proposals for multimedia composition and presentation, e.g., [HFK95, VS96], events correspond to temporal instances. We follow the latter understanding of the temporal aspect of events and consider events to be instantaneous.

Previous work in DBMS research on rules has focused on database triggers and alerters. Triggers test conditions in the database; if the database satisfies these conditions, then, the triggers fire actions. An alerter is a trigger that sends a message to a user or an application program if its conditions are satisfied.

Recent proposals for building rule sub-systems in DBMSs are reported in [S89]. The POSTGRES rule system, as originally proposed in [SHP88], uses a physical marking (locking) mechanism to ``lock'' data that may qualify trigger conditions.

In an effort to improve the execution efficiency of production rules, there has been research in using parallelism to solve the task of searching the database and identifying applicable rules and executing their actions. A motivation for executing rules in parallel is that one of the limiting factors for the speedup of the parallel match process is reported to be the (relatively small) number of updates made to the database, by the execution of a single rule. Thus, parallel execution of actions of different rules has the potential of affecting the parallel match phase as well [MT86, P89]. Srivastava, Hwang and Tan study the parallel execution of rules in [SHT90]. They identify two cases where parallelism can be used in database implementations of production rules: intra-phase and inter-phase parallelism. The former is achieved when many applicable actions of rules are allowed to run concurrently; the latter describes the concurrent processing of the different execution cycle phases. They provide some estimate of the benefits of concurrent execution, given that the rules execute in parallel. In [BKK87], MOBY, a distributed architecture to support the development of rule-based expert systems, in conjunction with a distributed database, is described. The focus of the research is to distribute the data so that parallel processors can be effectively utilized. The main problem that is studied is the use of database query processing and optimization techniques and incremental evaluation techniques, in order to solve the parallel match problem. Parallel execution of rules is only studied with respect to the effect it may have on the incremental computation techniques. The semantics of correct execution and the effect of executing actions on the conditions of other rules that may be executing in parallel, is not considered in this research.

Another project on rule execution that has had an impact on our work in executing rules as transactions is the HiPAC active database project [CM89]. Also in this work, (Condition, Action) or C-A pairs which are similar to rules, are triggered by events which could be updates to the database as well as external events. The concept of a transaction has been used to encapsulate the evaluation and the execution of the C-A pairs. Their research has produced some interesting results on the transaction boundaries within which these C-A pairs execute; simple transactions may execute a single Action, while more complex transactions may execute a sequence of such C-A pairs. We restrict our research to study the execution of simple transactions, where each transaction corresponds to a single production rule.

6 Conclusions

We presented an integrated approach for modeling, authoring and rendering IMDs. The model widely covers the issue of interactive scenarios and its constituents: events, spatiotemporal composition of presentation actions and scenario tuples. The result of IMD specification under this approach is a declarative script. We presented a scenario rendering scheme, based on a generic multi-threaded architecture that detects and evaluates events that occur in the presentation context and triggers the appropriate synchronized presentation actions. The rendering scheme has been implemented in a client-server system based on Java using the RMI (Remote Method Invocation) client-server communication protocol and the JMF (Java Media Framework) for handling multimedia objects. The system presents a promising approach for distributed interactive multimedia on the Internet and Intranets.

The salient features of the IMD modeling and subsequent authoring methodology are the following:

- Expressive set of temporal operators, enabling the definition of essentially any temporal composition expression, as they capture the causality of the temporal relationships among presentation intervals.
- Emphasis on Interaction: we provide a rich modeling and specification scheme for user- and intra-application interaction. The interactions are modeled in terms of events. We also provide a rich scheme for algebraic and temporal composition of events in order that arbitrarily complex events may be expressed. The proposed interactive scenario model is generic and open, and it can be utilized in several applications domains where interaction and spatiotemporal configuration is crucial. An example is virtual worlds, where there is intensive spatiotemporal configuration while the internal interactions (e.g. collisions of two moving objects etc.) could be represented as events.

- Declarative specification of the multimedia scenario, in terms of scenario tuples. In the scenario tuples we find autonomous synchronized presentation actions along with the event(s) that trigger or stop these actions. This approach makes feasible the storage of scenarios in databases, thus, exploiting all the well-known advantages of database support.

The salient features of the rendering system are:

- Platform independent design of a client-server system for IMDs. The system has strong distribution features since the media to be presented may reside at any http server.
- The choice of Java (along with accompanying technologies like RMI and JMF) as the implementation tool and the storage of media objects in http servers makes the design appealing for wide Internet usage. The clients are capable of presenting multiple scenarios simultaneously. The transformation of the IMDs, as defined in our system, into Java applets (which is trivial) makes the documents executable in any WWW browser.
- Lightweight and scalable design make the system applicable to a wide range of applications.
- Emphasis in interaction. Based on a rich IMD model, we have developed a robust mechanism for detection and evaluation of complex events as carriers of interaction and the corresponding synchronized media presentation algorithms.

The applicability of the approach presented in this paper is wide. The authoring tool, based on the sound theoretical background, provides the tools for robust and high level IMD creation. So far it has been used for educational material design and development. Though, the application domains include all the areas where interactive synchronized multimedia content is desirable. As it is apparent the requirements for such content is increasing along with the usage of WWW. As for the rendering mechanism, being platform independent and covering crucial issues, such as interaction, spatial and temporal synchronization, wide distribution features, makes it suitable for generic WWW enabled usage.

We have empirically evaluated both authoring and rendering tools. The first was used, as mentioned above, in learning material development. An initial familiarization with the authoring concepts was proved necessary. As for the rendering approach, it has been checked in a LAN and WAN basis. The results in the LAN case were satisfactory, while in the WAN case performance was tightly dependent on the network load.

The architecture presented here may be extended towards the following directions:

- *IMD spatiotemporal integrity*: we should also mention the issue of IMD integrity in terms of spatiotemporal constraints that arise from the specific features of media objects or from authors' requirements. Although our efforts have already come to initial results as regards the temporal domain [MPV99] we are going to further investigate the issue, especially the spatial aspects.
- *Provision of Quality of Sevice(QoS)*: provisions could be made to ensure the QoS. Admission control could be the first step towards this goal. It would be difficult to apply it to scenario requests, since, due to the interactive nature of scenarios, the bandwidth demands of a scenario are not fixed – they depend on user actions, but admission control could be used at every http server when a media object is requested. Moreover, the client could monitor the playback performance (or test it before the actual media object presentation), and refuse to present the video or sound if quality falls below a certain threshold. It is clear though, that due to the massively distributed architecture of the system, there is no apparent way of applying a centralized QoS control. In its present state, the system operates on a best-effort basis.

REFERENCES

[A83] J.F. Allen: "Maintaining Knowledge about Temporal Intervals". Communications of the ACM, Vol. 26, No. 11, November 1983, pp. 832-843.

[BKK87] Bein, J., King, R., and Kamel, N. MOBY: "Anarchitecture for distributed expert database systems". Proceedings of the Thirteenth Conference on Very Large Databases, Brighton, England 1987.

[BS96] G. Blakowski, R. Steinmetz, "A Media Synchronization Survey: Reference Model, Specification, and Case Studies", IEEE Journal on Selected Areas in Communications, vol 14, No. 1, Jan. 1996, pp. 5-35.

[B96] J. Buford, "Evaluating HyTime: An Examination and Implementation Experience". Proceedings of ACM Hypertext '96 Conference, 1996.

[Bul98] Dick C. A. Bulterman, Lynda Hardman, Jack Jansen, K. Sjoerd Mullender andLloyd Rutledge, "GRiNS: A GRaphical INterface for Creating and Playing SMIL Documents", Proceedings of Seventh International World Wide Web Conference(WWW7), April 1998.

[CPS98] K. S. Candan, P. Prabhakaran, V.S. Subrahmanian, “Retrieval Schedules based on resource availability and flexible presentation specifications”. ACM Multimedia Systems Journal, vol 6 (4), pp. 232-250, 1998.

[CM89] S. Chakravarthy, D. Mishra. “Rule management and evaluation: an active DBMS prospective”. In [S89].

[CM93] S. Chakravarthy, D. Mishra. “Snoop: An Expressive Event Specification Language For Active Databases”. Technical Report, UF-CIS-TR-93-00, University of Florida, 1993.

[DK95] A. Duda, C.Keramane. “Structured Temporal Composition of Multimedia Data”. CNRS-IMDG, Grenoble, France 1995.

[G94] S. Gatziou. “Events in Active Object-Oriented Database System”. Technical Report, ETH, Zurich, 1994

[GJS92] N.H. Gehani, H.V. Jagadish, O. Shmueli, “Composite Event Specification in an Active Databases: Model & Implementation”. Proceedings of VLDB Conference, 1992, pp. 327-338.

[H96] M. Handl, "A New Multimedia Synchronization model". IEEE Journal on Selected Areas in Communications, vol 14, No. 1, Jan. 1996, pp. 73-83.

[HCD98] I. Herman, N. Correia, D. Duce et al, “A standard model for multimedia synchronization: PREMO synchronization objects”. ACM Multimedia Systems Journal, vol 6(2), pp. 88-101, 1998.

[HFK95] N. Hirzalla, B. Falchuk, A. Karmouch. “A Temporal Model for Interactive Multimedia Scenarios”. IEEE Multimedia, Fall 1995, pp. 24-31.

[IDG94] M. Iino, Y.F. Day, A. Ghafoor. “An Object Oriented Model for Spatiotemporal Synchronization of Multimedia Information”. In Proc. IEEE Int. Conf. Multimedia Computing and Systems, Boston MA, 1994, pp. 110-119.

[ISO92] International Standard Organization. “Hypermedia/Time-based Document Structuring Language (HyTime)”, ISO/IEC IS 10744, April 1992.

[ISO93] ISO/IEC. JTC 1/SC 29. “Coded Representation of Multimedia and Hypermedia Information Objects (MHEG)”, Part I, Committee Draft 13522-1, June 1993. ISO/IEC 10031.

[J97a] Java-Remote Method Invocation, available at: http://java.sun.com:81/marketing/collateral/rmi_ds.html

[J97b] Java–Media Framework available at: http://www.javasoft.com/products/java-media/jmf/

[KE96] A. Karmouch, J. Emery. “A playback Schedule Model for Multimedia Documents”, IEEE Multimedia, v3(1), pp. 50-63, 1996

[KSV99] I. Kostalas, T.Sellis, M. Vazirgiannis, “Spatiotemporal specification & verification of multimedia scenarios”, in the proceedings of the IFIP-DS-8 Conference on Multimedia Semantics, January 1999.

[LG93] T. Little, A. Ghafoor. “Interval-Based Conceptual Models for Time-Dependent Multimedia Data”. IEEE Transactions on Data and Knowledge Engineering, Vol. 5, No. 4, August 93, pp. 551-563.

[MT86] Miranker,D.P.TREAT: “A new and efficient match algorithm for AI production systems”. Ph.D. thesis, Department of Computer Science, Columbia University,New York (1986).

[MPV99] I. Mirbel, B. Pernici, M. Vazirgiannis, "Temporal Integrity Constraints in Interactive Multimedia Documents", in the proceedings of IEEE-International Conference on Multimedia Computing and Systems (ICMCS’99), Florence, Italy, June 1999.

[P89] Pasik, A.J. “A methodology for programming production systems and its implications on parallelism”. Ph.D. thesis, Department of Computer Science, Columbia University,New York (1989).

[RFC2205] R. Braden et al. “Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification”. RFC 2205, September 1997.

[S89] T. Sellis, Editor. “Special issue on rule management and processing in expert database systems”. SIGMOD Record (18) 3 (1989).

[SHT90] J. Srivastava, K.-W. Hwang, and J.N. Tan. “Parallelism in database production systems”. Proceedings of the 6th IEEE International Conference on Data Engineering (1990).

[SKD96] J. Schnepf, A. Kosntan, D.H. Du. “Doing FLIPS: FLexible Interactive Presentation Synchronisation”, IEEE Journal on Selected Areas in Communications, Vol. 14, No . 1, January 1996, pp. 114-125.

[SW95] G. Schloss, M. Wynblatt. “Providing definition and temporal structure from multimedia data”, Multimedia Systems Journal, Vol. 3, 1995, pp. 264-277.

[SMI98] Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C, Proposed Recommendation.http://www.w3.org/TR/REC-smil/

[SHP88] M. Stonebraker, E.N. Hanson and S. Potamianos. “The POSTGRES rule manager”. IEEE Transactions on SoftwareEngineering (14) 7 (1988).

[TKWPC96] H. Thimm, Wolfgang Klas, Jonathan Walpole, Calton Pu, Crispin Cowan. “Managing Adaptive Presentation Executions in Distributed Multimedia Database Systems”. In: Proceedings of International Workshop on Multimedia Database Systems, Blue Mountain Lake, NY, 1995.

[V96] M. Vazirgiannis, "Multimedia Data Base Object and Application Modelling Issues and an Object Oriented Model" in the book "Multimedia Database Systems: Design and Implementation Strategies " (editors Kingsley C. Nwosu, Bhavani Thuraisingham and P. Bruce Berra), Kluwer Academic Publishers, 1996, Pages: 208-250

[VB97] M. Vazirgiannis, S. Boll, "Events In Interactive Multimedia Applications: Modeling And Implementation Design", in the proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS'97), June 1997, Ottawa, Canada.
[VM93] M. Vazirgiannis, C. Mourlas, "An object Oriented Model for Interactive Multimedia Applications", The Computer Journal, British Computer Society, vol. 36(1), 1/1993.
[VS96] M. Vazirgiannis, T. Sellis. "Event and Action Representation and Composition for Multimedia Application Scenario Modelling". In the procedings of ERCIM Workshop on Interactive Distributed Multimedia Systems and Services, BERLIN, 3/1996.
[VTS96] M. Vazirgiannis, Y. Theodoridis, T. Sellis. "Spatio Temporal Composition in Multimedia Applications". In: Proceedings of the IEEE - ICSE '96 International Workshop on Multimedia Software Development - BERLIN, 3/1996.
[VTS98] M. Vazirgiannis, Y. Theodoridis, T. Sellis, "Spatiotemporal Composition and Indexing for Large Multimedia Applications" , ACM/Springer-Verlag Multimedia Systems Journal, 1997
[Yu97] Jin Yu, Yuanyuan Xiang, "Hypermedia Presentation and Authoring System", in proceedings of the 6th International World Wide Web Conference, pages 153-164, April 1997

APPENDIX A: A sample IMD Scenario

In this appendix we refer to a sample IMD involving complex interaction. The presentation is related to ancient Olympic Games. We have a set of related media objects ("actors") that we want to use in this presentation. The actors are presented according to spatial and temporal transformations and specifications according to authors requirements. This conveys the spatial and temporal coordinates of the objects in the presentation context as well as their transformations under which they would be included in the presentation. These transformations are formally described in [V96]. The actors' specifications and transformations appear in Table 1. As mentioned above the carriers of interaction in our model are the events (atomic or complex). Hereafter we refer to the events that the sample presentation will detect and consume. They are the following:

- DoubleClick: raised when the user makes a Mouse DoubleClick
- KeyEsc: raised when the user presses the key "Escape"
- IntroStop: raised each time the audio clip INTRO ends its playback
- ExitEvent: raised each time the user presses the button EXITBTN
- TIMEINST: raised each time the timer TIMER1 reaches the 50th second.
- AppTimerEvent: raised each time the ApplicationTimer reaches the time 02min and 30 seconds.
- ACDSoundStop: raised each time the audio clip ACDDM stops
- NextBtnClick: raised each time the user presses the button NEXTBTN

The events formally defined appear in Table 2.
Having the media objects available we may build the application for the folowing scenario:

"The application starts (event StartApp) with presenation of button LABELBTN immediatelly followed by the audio clip INTRO. After 3 seconds the image IMG2_1 followed by IMG2_2 after 2 seconds. After 2 more seconds the image IMG1_2 is presented at position and after 2 seconds IMG2_1, IMG2_2 and IMG1_2 stop their presentation (i.e. dissapear from the screen) while after 1 second the video clip RUNS starts. This sequence of presentation actions may be interrupted whenever one of the events: , _DoubleClick, _KeyEsc, _IntroStop occurs.

Another set of presentations ("Stage2A")starts when the event _IntroStop is raised. The presentation actions that take place are: presentation of image IMG1_3 in parallel with audio clip ACDDM (when the clip ends the image disappears). At the same time, two seconds after timer TIMER1 (that started when Stage 2A started) expires the text INFOTXT is presented.
The next set of media presentations ("Stage 2B") is initiated when the sequence of events _IntroStop and _ACDSoundStop occurs. During Stage2b the video clip KAVALAR starts playback while the buttons NEXTBTN and EXITBTN are presented. The presentation actions are interrupted when any of the events _TIMEINST and _NextBtnClick occurs. The end of Stage2b raises the synchronization event _e1.

The following set of media presentations ("Stage3") start when any two of the events _e1, _NextBtnClick, _TIMEINST occur. During Stage3 the text INFOTXT disappears, just after that the text YMNOS appears and the audio clip FLUTE starts while two seconds after image IMG1_1 and IMG3_1 appear. Three seconds after the EXITBTN appears. The last part of the scenario handles the presentation termination which will occur when either the _ExitEvent or the _AppTimerEvent are raised" .

The above scenario is represented by the scenario tuples appearing in Table 3. In the scenario tuples for simplicity reasons we have excluded the spatial relationships information, which is to a great extent profound in the definition of actors.

All the specifications that appear in the following tables are results of the authoring procedures.

ACTOR LABELBTN Type of actor = LABEL Caption = OLYMPIC GAMES Height = 120 Width = 333 Xcoord = 100 Ycoord = 0 Transparent = True FontFace = HellasTimes FontSize = 28 Color = 255,255,0 FontStyle = Regular	ACTOR INTRO Type of actor = WAVE FileName = INTRO.WAV PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Start = 0 Duration = 46874 Volume = 100 Looping = No Direction = Forward Effect = none	ACTOR IMG1_1 Type of actor = GIF FileName = Callisto.gif PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Height = 100 Width = 166 Xcoord = 366 Ycoord = 33 Layer = 1 PreEffect = none PostEffect = none
ACTOR IMG1_2 Type of actor = GIF FileName = Marsice.gif PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Height = 120 Width = 166 Xcoord = 366 Ycoord = 93 Layer = 1 PreEffect = none PostEffect = none	ACTOR IMG1_3 Type of actor = GIF FileName = Callisto.gif PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Height = 233 Width = 200 Xcoord = 33 Ycoord = 66 Layer = 1 PreEffect = none PostEffect = none	ACTOR RUNS Type of actor = AVI FileName = OPLODROM.AVI PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Start = 0 Duration = 1667 Scale Factor = 3 Volume = 70 Looping = Yes Direction = Forward Height = 200 Width = 233 Xcoord = 166 Ycoord = 133
ACTOR IMG2_1 Type of actor = GIF FileName = Marsice.gif PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Height = 233 Width = 166 Xcoord = 33 Ycoord = 93 Layer = 1 PreEffect = none PostEffect = none	ACTOR IMG2_2 Type of actor = GIF FileName = Callisto.gif PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Height = 133 Width = 133 Xcoord = 200 Ycoord = 177 Layer = 1 PreEffect = none PostEffect = none	ACTOR KAVALAR Type of actor = AVI FileName = KAVALAR.AVI PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Start = 291 Duration = 1709 Scale Factor = 3 Volume = 70 Looping = Yes Direction = Forward Height = 200 Width = 166 Xcoord = 33 Ycoord = 66
ACTOR INFOTXT Type of actor = TXT FileName = OLYMPIC1.TXT PathName = htttp://www.ddi.uoa.gr//~stud0627//recources Height = 266 Width = 266 Xcoord = 333 Ycoord = 33 Transparent = True Layer = 1 BorderStyle = scrolling FontFace = HellasArial FontSize = 12 Color = 255,0,0 FontStyle = Regular	ACTOR NEXTBTN Type of actor = BUTTON Caption = Next Height = 30 Width = 100 Xcoord = 333 Ycoord = 366 Transparent = False FontFace = MS Sans Serif FontSize = 14 Color = 255,0,0 FontStyle = Bold	ACTOR EXITBTN Type of actor = BUTTON Caption = Exit Height = 33 Width = 100 Xcoord = 466 Ycoord = 366 Transparent = False FontFace = MS Sans Serif FontSize = 14 Color = 255,0,0 FontStyle = Bold
ACTOR FLUTE Type of actor = WAVE FileName = S_FLUTE.WAV PathName = htttp://www.ddi.uoa.gr//~stud0627//reco	ACTOR IMG3_1 Type of actor = GIF FileName = Marsice.gif PathName = htttp://www.ddi.uoa.gr//~stud0627//reco	ACTOR YMNOS Type of actor = TXT FileName = YMNOS.TXT PathName = htttp://www.ddi.uoa.gr//~stud0627//r

urces Start = 0 Duration = 8734 Volume = 90 Looping = Yes Direction = Forward Effect = none	urces Height = 100 Width = 166 Xcoord = 366 Ycoord = 200 Layer = 1 PreEffect = none PostEffect = none	ecources Height = 386 Width = 233 Xcoord = 200 Ycoord = 333 Transparent = True Layer = 1 BorderStyle = raised FontFace = HellasArc FontSize = 10 Color = 128,0,128 FontStyle = Italic
ACTOR TIMER1 Type of actor = TIMER	ACTOR ACDDM Type of actor = WAVE FileName = ACDMC2M1.WAV PathName = htttp://www.ddi.uoa.gr//~stud 0627//recources Start = 0 Duration = 10961 Volume = 70 Looping = No Direction = Forward Effect = none	

Table A.1: Actors List

EVENT _DoubleClick Subject = Mouse Action = DoubleClick	EVENT _KeyEsc Subject = Keyboard Action = Escape	EVENT _IntroStop Subject = INTRO Action = Stop
EVENT _ExitEvent Subject = EXITBTN Action = Click	EVENT _TIMEINST Subject = TIMER1 Action = 50	EVENT _AppTimerEvent Subject = ApplicationTimer Action = 150
EVENT _ACDSoundStop Subject = ACDDM Action = Stop	EVENT _NextBtnClick Subject = NEXTBTN Action = Click	

Table A.2. Events list

TUPLE ExitTuple Start Event = _ExitEvent	_AppTimerEvent Stop Event = Action List = ExitApplication Start Synch Event = none Stop Synch Event = none
TUPLE Stage1 Start Event = StartApp Stop Event = ANY(1;_DoubleClick;_KeyEsc;_IntroStop) Action List = LABELBTN> 0 INTRO> 3 IMG2_1> 2 IMG2_2> 2 IMG1_2> 2 IMG2_1< 0 IMG2_2< 0 IMG1_2< 1 RUNS> Start Synch Event = none Stop Synch Event = none
TUPLE Stage2A Start Event = _IntroStop Stop Event = Action List = (IMG1_3 /\ ACDDM) ; ($> 0 TIMER1> 2 INFOTXT>) Start Synch Event = none Stop Synch Event = none
TUPLE Stage2B Start Event = SEQ(_IntroStop;_ACDSoundStop) Stop Event = ANY(1;_TIMEINST;_NextBtnClick) Action List = KAVALAR> 0 NEXTBTN> 0 EXITBTN> Start Synch Event = none Stop Synch Event = _el
TUPLE Stage3 Start Event = ANY(2;_el;_NextBtnClick;_TIMEINST) Stop Event = Action List = $> 0 INFOTXT< 0 FLUTE> 0 YMNOS> 2 IMG1_1> 0 IMG3_1> 3 EXITBTN> Start Synch Event = none Stop Synch Event = none

Table A.3. The scenario tuples