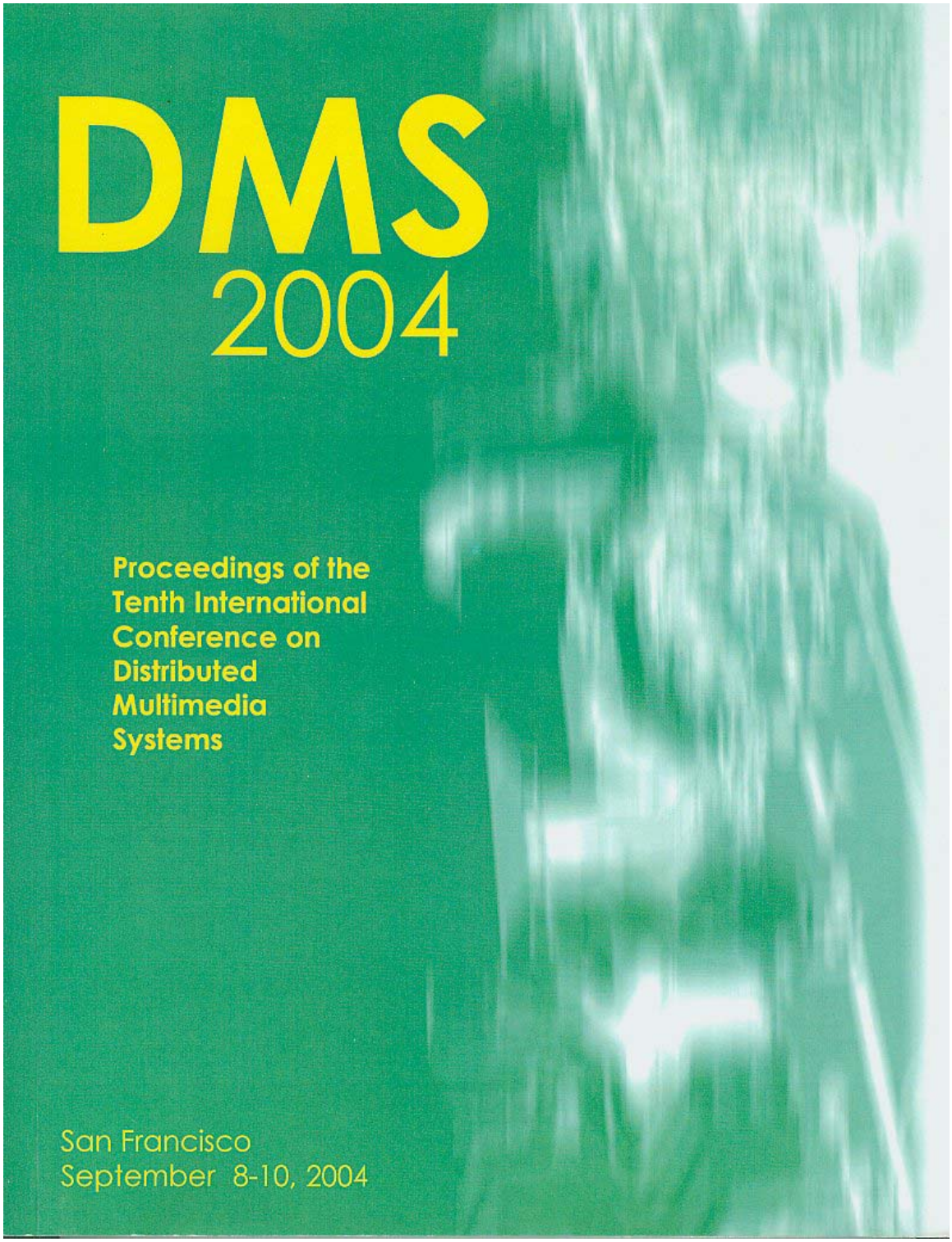
DMS
2004
Proceedings of the
Tenth International
Conference on
Distributed
Multimedia
Systems
San Francisco
September 8-10, 2004

TANDEM – Transmitting Asynchronous Non Deterministic and Deterministic Events in Multimedia Systems over the Internet

Angela Guercio and Arvind K. Bansal
Department of Computer Science
Kent State University, Kent, OH 44242
guercioa@hiram.edu and arvind@cs.kent.edu
Tel. 330-672 9035 Fax. 330-672 7824

Abstract

Internet is being used more pervasively for multimedia retrieval, multimedia transmission and rendering. However, little work has gone on the Internet based multimedia modeling integrating user interaction, asynchronous and non-deterministic multimedia events, multiple multimedia streams, and their synchronization. In this paper, we describe a new and novel XML based multimedia language 'TANDEM' which supports the integration of user interaction, asynchronous non deterministic multimedia events, multiple multimedia streams, and their synchronization. More specifically, we describe 'event constructs', 'stream grouping constructs', 'transformation constructs', and 'synchronization constructs'. We illustrate the constructs with relevant examples.

Keywords: asynchronous, event, Internet, language, modeling, multimedia, nondeterministic, synchronization

1. Introduction

As the Internet becomes pervasive, the multimedia knowledge base on the Internet will become source for the Internet based multimedia modeling application based upon user interaction. For an effective use of distributed multimedia data over the Web, we need to develop a series of tools and languages that can help programmers to create such applications.

Current multimedia languages [9] are evolving, and are limited in their capability to model real world phenomenon which needs integration of multimedia reactivity, asynchronous events, computability, event-based triggering, dynamic altering of multimedia attributes, and synchronization of multiple streams. Languages for modeling distributed multimedia systems must support asynchronous events, loose ordering of events, and automated dilation of time-scale in a group of streams to preserve synchronization.

This paper introduces TANDEM — an XML based distributed multimedia language for Transmitting Asynchronous Non-deterministic and Deterministic Events in Multimedia systems. The language supports the development of distributed multimedia systems which integrate deterministic and asynchronous non-deterministic events over the Internet. The language is based on a conceptual model that has a trigger — mechanism which controls the reaction to multimedia content in multimedia streams — as the event generator which instructs the transformers for the appropriate reaction. A trigger accepts multimedia streams (both periodic and aperiodic) as input along with the associated media contents, and generates one or more event. The language provides constraints-based synchronization facilities with dynamic scaling for groups of multimedia streams, which have a clock associated with them to control their synchronization. Context-related reactions to situations are detected by an active repository, as well as reactions to the external world. The use of a persistent repository takes care of partial conditions and the change in the order of nondeterministic conditions since the truthfulness of partial and nondeterministic conditions can be archived in the active repository.

The major contributions of this paper are:

(1) The language allows the integration of asynchronous events, deterministic multimedia events, and multimedia event based triggering commands.

(2) The language supports temporal and spatial synchronization of the complex media objects.

(3) The language separates and integrates five major components of distributed multimedia reactive systems: triggers, active repository, transformers, grouping of media streams, and synchronization.

(4) The language allows dynamic scaling of the multimedia time base for implicit synchronization

(5) The language incorporates commands that interrogate the active repository for partial conditions verification as well as pattern detection, oblivious data and context-based conditions.

The paper is organized as follow. Section 2 describes briefly related concepts in distributed reactive multimedia systems. Section 3 describes the conceptual model for TANDEM. Section 4 describes the constructs for multimedia events, stream grouping for synchronization, synchronization and transformation constructs. Section 5 presents related works. The last section concludes the paper. Due to the space limitation, the grammar for the language and a description of a major nontrivial application will be presented elsewhere.

2. Distributed Reactive Multimedia Systems

Multimedia systems provide interaction involving text, graphics, audio and video. The interaction is obtained via multimedia streams and aperiodic signals. A multimedia stream is a sequence of nested tuples and data values. A multimedia stream can be:

(1) a *continuous stream* as produced by sensors
(2) a *periodic stream* where data is associated with a periodic signal, or
(3) an *aperiodic stream* where the data is associated with an aperiodic signal generated by an event or external interaction.

A *multimedia stream S* has two components: *attribute-set* and *data*. Three attributes *periodic* or *aperiodic*, number of data elements per unit time, and type of data (such as *audio* or *video* or *music or audiovisual* etc.) are essential. Other attributes are specific to the streams, and vary with different types of multimedia streams.

Example 1: The data for the audio stream is a sequence of sampled packets with the attributes: (a_0=periodic, a_1=audio, a_2 = 44100 samples/second, a_3 = no. of channels = 4, a_4 = 16 bits per sample, a_5 = media length, …).

Each multimedia media stream has its own clock which is synchronized to a *common*, and is played according to the *playback rate* R of the media type.

In distributed reactive multimedia system [2], a number of multimedia streams and aperiodic signals are produced in one location (local or remote) and are consumed in another location. Multiple streams are synchronized with each other, are transformed, and the system reacts to asynchronous events. An aperiodic signal interacts with other streams or signals, transform a stream (or group of streams), and trigger another chain of events. Media types react to external stimuli (user intervention) or their own content, or to some other media type content. The repetition of the same actions does not guarantee the same reaction due to the change in context, past events, or the order of events.

Distributed reactive multimedia systems must be able to react when certain conditions are met. The reaction of the system consists of the generation of one or more events that "respond" appropriately to the presence of some previous phenomena, which consists in the satisfaction of a set of Boolean conditions.

In distributed reactive multimedia system interactions with remote locations is a necessity. Consider an on-line conference in which participants draw on a shared whiteboard object. The drawing must be visible to all the participants immediately. Actions, which do not directly require media stream's attributes modification, should be able to deal with network issues, mobile users, security and alarm exceptions and resource managements.

3. The Conceptual Model of TANDEM

The application module of TANDEM is based on the conceptual model for the creation of distributed reactive multimedia systems introduced in [5]. The distributed multimedia application is modeled as reaction graphs (see Figure 1) as follows:

(i) Media generation points are modeled as *sources*, and a media rendering (or archival) points are modeled as *sinks*. Sources and sinks can be local or remote. A *sink* is a URI where the multimedia stream is rendered or stored.
(ii) *Transformer* nodes (see Figure 2) apply transformation functions to modify the attributes of the multimedia stream. For example, transformers change the rendering rate of a group of streams, multiplex streams, or reduce the number of channels of a video for rendering.
(iii) *Triggers* provide a general mechanism for actions after a set of Boolean conditions are met. Trigger nodes control the media streams and initiate reaction to the streams. A trigger is associated with sets of multimedia stream groups and a set of conditions. Triggers activate events in response to the satisfaction of a set of conditions. A trigger reaction includes monitoring of external sensors, transforming the attributes of multimedia streams, redirecting a stream to a different destination, starting a new thread of computation, or a cascade of triggers and events. A trigger reaction might require a computation which involves the input streams; in that case the input streams are pre-transformed by one or more transformer modules.
(iv) *Active repository* nodes are associated with triggers. Active repository samples and analyzes the media content for the required conditions, and transmits the outcome to the triggers. An active repository detects and archives *partial conditions* to match the conditions in loose order of occurrence.

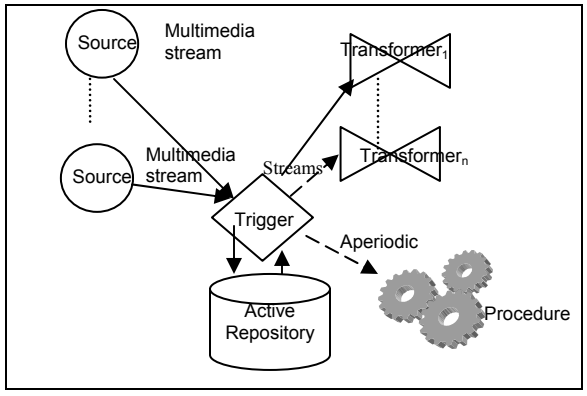

Figure 1. The conceptual execution model

We distinguish two types of triggers: *event-based triggers* and *periodic triggers*. Event-based triggers fire when some constraints (or conditions) are satisfied. For example, the alarm activation of a surveillance camera fires a trigger when a human figure is detected. Periodic triggers fire periodically. For example, a biologist capturing the blooming of a flower will capture and transmit pictures periodically. Triggers are also classified as *continuous* or *discrete*. *Continuous* triggers do not require resetting, while *discrete* triggers are reset every time they go off. Triggers are associated with zero or more streams and aperiodic signals. The trigger communicates with an active repository to archive and retrieve partial conditions and allow nondeterministic order of conditions in multiple streams If the reaction requires a computation which does not involve the input streams, the trigger generates an aperiodic signal which causes a procedure-call associated with the activated event, shown by the gears.

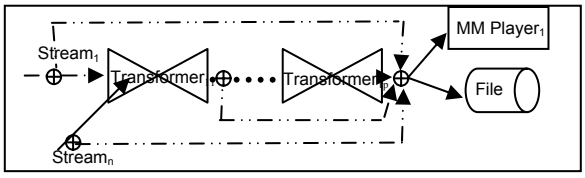

Figure 2. Application of transformers

Groups are used to perform operations on sets of related multimedia streams. Operation on multimedia streams can be either *'isolated'* or *'joint'*. Isolated operations do not affect individual media streams in a group. Joint operations affect every member of the group. Synchronization of streams is a joint operation. Multiple groups can be associated with a trigger. Although a part of triggers, *aperiodic* signals are not part of groups.

Together with the input signals, i.e. groups of periodic multimedia streams (audio, video, text, audiovisual etc.) or aperiodic signals, and active respository, the trigger forms the basic building block to generate an *event*. The generated events, represented by the dotted links in the above figure, perform the reaction to the specific *conditions* verifications. Event definitions in the trigger define all the conditions that must be verified for the event to occur. The destination of an event can be a *transformer*, a *procedure, or a sink.* A *procedure* is activated when computations (other than stream transformation) are required. Synchronization is required after detecting a required pattern(s) in the active repository. The active repository signals the trigger for the verification of the *partial conditions*, and the action.

4. TANDEM: Framework and Constructs

In this section we introduce generic constructs. Since TANDEM is an Internet language the constructs follow the XML style; however, constructs are generic.

A distributed reactive multimedia application is constructed by specifying the sources, the sinks, the triggers, the transformers, the active repository, and the events. We first define the streams involved in the application. The streams will be input to one or more triggers. Each trigger will contain the description of the groups that participate in the trigger, the loop constructs that we apply to the input streams, the events that the trigger will handle and the destinations of those events. We describe the destinations of each event in the trigger. The outline of an application framework is:

```
<application ...>
  # we describe all the media streams
  <mediastream ... > # mediastream 1 </mediastream>
  ...
  <mediastream... > # mediastream M </mediastream>
  # we drescribe all the triggers
  <trigger ...>
    # define all the loops involving media streams
    <loop ...> ...#loop1 </loop>
    ...
    <loop ...> ... #loopN </loop>
    # define all the groups
    <group ...> . #group 1 </group>
    ...
    <group ...> . #group G </group>
    #define events, conditions and trigger destinations
    <event ...> . #event 1 </event>
    ...
    <event ...> .# event E </event>
    #describe transformers, actions and destinations
    <transformer ...> . #transformer 1 </transformer>
    ...
    <transformer ...> . #transformer S </transformer>
    #describe all the procedures required
    <procedure ...> .#procedure1 </procedure>
    ...
    <procedure ...> .#procedure P </procedure>
  </trigger>
  ...
  <trigger>... #trigger T </trigger>
</application>
```

Figure 3. An outline of application framework

The definition of the multimedia streams in the trigger provides a way to collect information about all the sources (remote or local) needed for the application. For example, a remote source that is sending an audio at a 44,100 samples/sec. rate collects the data and values of the attributes as follows:

```
<media_stream name = "mm1">
  <source name = "source1" URI = "192.168.2.102" />
  <type> <audio name = "audio.wav"
       samples_per_sec = "44100" no_of_channels = "2"
       bits_per_sample = "16"
       scaling_constraint = "[0.5, 2]" />
  </type>
</media_stream>
```

Repetition of streams is performed in the *loop construct*. The construct contains the name of the loop and defines the elements which participate in the loop. For example, the following construct defines a repetition of stream *mm1* 3 times.

```
<loop name = "mm1loop" times = "3">
        <loop_element name = "mm1" />  </loop>
```

Nested loops are defined by referencing the name of a predefined loop. The following code repeats *mm1loop* five times.

```
<loop name = "nestedloop" times = “5”>
        <loop_element name = “mm2” />
        <loop_element name = "mm1loop" /> </loop>
```

4.1 Event constructs

An event is generated when specific conditions are satisfied. An event has a destination according to the action that the event must perform. The events are either tightly integrated with the spatial and temporal constraints or they may loosely coupled. In a tightly integrated constraint the order of the events is very specific and the temporal and spatial constraints are strictly followed. Loosely coupled events may have non-deterministic order of events with more relaxed constraints.

In the following code fragment, the event "start_video" starts a video clip if all the partial conditions are satisfied. The partial conditions verify the presence of two aperiodic signals: user's right-clicking and the end of another video clip. After both the conditions are satisfied the event is generated. The destination of the event is the "transformer-1", which takes as input data stream from the source = "video1".

```
<application name = "example">
...
   <event name = "start_video"
        Start = "immediately" priority = "1"... >
      <condition name = "cond1"
           signal_type = "rightclick"
           presence = "present" ... />
      <condition cond_name = "cond2"
           signal_type = "video2_end"
           presence = "present" ... />
      <destination name = "transformer1"
           Source = "video1"... />
   </event>
...
</application>
```

In TANDEM, the generated events can involve multiple sources or multiple groups of multimedia streams. A generated event is sent to one or more destination(s). Remote location interaction can be performed by using multiple remote destinations. The ordering of the events in an application is handled by an optional priority value. The priority, when specified, guarantees the partial order of the events.

4.2 Group constructs

Grouping clusters one or more media streams or groups for synchronization. The groups are both *dynamic* and *hierarchical*. Groups are an elegant and efficient way to specify synchronization on multiple streams as follows:

```
<group name = "soprano">
  <member name = "mm1"/> <member name = "mm2"/>
</group>
```

The construct creates the group *"soprano"* which groups two media streams, *mm1* and *mm2*. Groups can be modified dynamically by a transformer using group actions as follows:

(i) *ungroup*, to separate an existing group,
(ii) *add_group*, to add elements to a group
(iii) *delete_group*, to eliminate elements from a group
(iv) *regroup*, to incorporate elements into a new group

In the example described below, the transformer “*ungroup-soprano”* ungroups the elements *mm1* and *mm2* of the group *soprano*. After ungrouping the group *soprano* does not exist. Regrouping is needed for group reconstruction before further use.

```
<transformer name = "ungroup-soprano">
   <action >
        <ungroup > <elements group = "soprano"/>
        </ungroup>
   </action>
   <destination name = "player1" />
</transformer>
```

4.3 Synchronization constructs

Synchronization in multimedia requires the ability to relate the elements involved in the multimedia application both spatially and temporally. The synchronization specifies how those elements will to be presented in a specific spatial or temporal order on the rendering device. The required synchronization can be influenced by the events caused by the user interacting with the multimedia system.

The synchronization constructs (i) provide synchronization in the presence of external constraints, such as user interactions, (ii) provide synchronization in the presence of system environmental constraints, such as network traffic or resource availability, and (iii) provide synchronization for user defined media groups.

A *trigger* activates an event every time specific conditions are satisfied. The activation can be started immediately or delayed. In the construct (1) the trigger starts the event immediately, in the construct (2) the

delay is 9 units. The delay can be determined by a computation as shown in the construct (3).

```
<event name = "start_video" start = "0" > ... </event>    (1)

<event name = "start_video" start = "9" > ... </event>   (2)

<event name = "synch&start" start = "0"
     var_delay = expression > ... </event>                (3)
```

An event might need to invoke a computation. For example, an event might be delayed depending on conditions of the network traffic. An event "invoke_compute" is going to start a procedure "compute_delay" which checks for the network conditions and calculates the appropriate delay. This is done within a period trigger as shown below:

```
<event name = "invoke_compute" start = "0">

#test the presence of the signal "procedure_start"

   <condition name = "cond2" signal_type =
        "procedure_start" presence = "present"/>
   <destination name = "compute_delay" />
</event>

# check and compute the network delay

<procedure name = "compute_delay" start = "0">
    <parameters> <param name = "delay" type = "int"
          Mode = "out"/> </parameters>
</procedure>
```

Whenever synchronization is required an event must be generated to activate the appropriate reaction. The trigger identifies the specific transformer, and the synchronization is activated. The transformer must contain the synchronization actions to be performed. Since transformers alter the attributes of multimedia streams both spatially and temporally, the synchronized actions must distinguish between spatial and temporal synchronization. For example, consider two media streams are played in *parallel* such that their *start together and end together* match. Then the trigger will start an event that will have as destination a transformer that performs the synchronization.

In the following construct, the transformer starts the two streams *mm1* and *mm2* in parallel: *mm2* will start 3 units later than *mm1* and will end at the same time. The stretch required to perform the synchronization must be compatible with the previously defined scaling constraints. If stream scaling constraints are not verified, no modifications will be performed on the stream. The scaling type construct 'Stretch' causes the speed up or the slow down of the playback rate of the media object.

```
<trigger name = "trig1" ...>
...
 <event name = "parallel_start"...>
 ...
    <destination name = "transf1"/>
 </event>
 ...
<transformer name = "transf1">
 <action >
  < synchronize >
     <temporal type = "start&end" scaling = "stretch"
         Reference = "mm1" start = "0" end = "0">
        <elements name = "mm2" diffstart = "3"
            Diffend = "0"/>
     </temporal>
  </synchronize>
 </action>
 <destination name = "player1" />
</transformer>
...
</trigger>
```

If the parallel start is applied to a group of streams, the group elements are related synchronously to the reference stream. In the code fragment below, all the media streams of *group1* start 3 units later than *mm1* and terminate at the same time, while all the media streams of *group2* start after 2 units and terminate 2 units earlier with respect to the stream *mm1*.

```
<synchronize>
   <temporal type = "start&end" scaling = "stretch"
         Reference = "mm1" start = "0" end = "0" />
      <elements name = "group1" diffstart = "3"
         Diffend = "0" />
      <elements name = "group2" diffstart = "2"
        Diffend = "-2" />
   </temporal>
</synchronize>
```

Synchronization actions involving spatial constraints are computed by giving the relative values. The following code locates all the elements of *group1*, on the X-axis, 3 points after the X-position of *mm1*. Synchronization is handled by the synch_type = "seq_start" as follows.

```
<synchronize>
   <spatial type = "start&end" scaling = "stretch"
       Alignment = "upper_left">
     <elements name = "mm1" diffstartX = "0"
       diffstartY = "0" diffendX = "0" diffendY = "0"/>
     <elements name = "group1" diffstartX = "3"
       diffstartY = "0" diffendX = "3" diffendY = "0"/>
   </spatial>
</synchronize>
```

7. Related Works

The research involving synchronized multimedia streams [8] and the use of high level distributed multimedia language constructs such as event triggering and synchronization constructs for flexible Internet based modeling is evolving rapidly after the advent of the Internet. The languages like SMIL [9], TAOML [1],

VRML [10] and its XML based variants, synchronous language Estrel [3], and distributed multimedia languages concerned with QoS (Quality of Service) [6] have different aspects of multimedia modeling, synchronization constructs and event based constructs.

SMIL [9] models concurrent multimedia streams by synchronizing start and end of the streaming at a specific point of time relative to other streams, and supports spatial synchronization and placement of media streams. However, there is no comprehensive stream group construct in SMIL, and SMIL also does not support frame level synchronization due to the lack of synchronization of periodic signals. The event model (used primarily for user interaction) is independent of the timing model (used for played back). If no events are defined by a host language, event-timing is effectively omitted [9]. For example, if a video is started by the left-click of the mouse and is ended by the right-click of the mouse, in SMIL we cannot guarantee that the left-click followed by the right click results in the media starting and stopping. SMIL also does not support nondeterministic order of events and partial matching of conditions.

VRML [10] and its XML based variants support the notion of events and events triggering another events, triggering computation, and grouping of multimedia components. The events can be modified dynamically in VRML However, VRML also does not support synchronization of periodic stream at the Frame level, and does not support nondeterministic order of conditions or archiving partial conditions.

TAOML [1] has limited expression capability. For example, nested loop or dynamic grouping is available neither in TAOML nor in SMIL. Other languages, such as HQML [6] have focused their attention to the quality of service capabilities; therefore their synchronization abilities are very limited.

ESTREL [3] is a synchronous deterministic hardware modeling language. Integrated with distributed multimedia sources and sinks constructs Estrel constructs can be used to model multimedia objects and periodic multimedia streams, and triggering a chain of multimedia events. However, Estrel is not an Internet based language, and does not support explicitly grouping of periodic streams and nondetermistic order of events or storing of partial conditions.

6. Conclusions

In this paper, we have described a distributed multimedia modeling language TANDEM that integrates deterministic, non-deterministic events, asynchronous events, and spatio-temporal synchronization. The language is based upon a novel model [5] developed by the authors which uses a persistent active repository. The active repository supports pattern-based matching and content-based analysis of media streams. The use of the persistent active repository allows us to store partial conditions and relax the order of events. The language has five distinct types of constructs, namely, multimedia definition constructs, group constructs, trigger constructs, synchronization constructs, and transformer constructs. We support multiple stream synchronization using group constructs, support aperiodic signals for user interaction, and continuous streams for sensor information. Trigger constructs are used either to invoke media transformers that alter the attributes of media streams or invoke a procedure call for computation. The language is in advanced stage of implementation.

References

[1] T. Arndt, S.K. Chang, A. Guercio, “Formal Specification and Prototyping of Multimedia Applications,” *Int. Journal of SEKE.*, vol.10, no.4, pp.377-409, 2000.

[2] J. Bacon et al., "Generic Support for Distributed Applications", IEEE *Computer*, pp. 2-10, March 2000.

[3] G. Barry and G. Gonthier, “The ESTREL Synchronous Programming Language: Design, Semantics, Implementation,” Science of Computer Programming, 19(2), 1992

[4] X. Gu, K. Nahrstedt, "An Event-Driven, User-Centric, QoS-aware Midleware Framework for Ubiquitous Multimedia Applications", *Proc. of ACM Multimedia*, Ottawa, Oct. 2001.

[5] A. Guercio, A. K. Bansal, "A Model for Integrating Deterministic and Asynchronous Events in Reactive Multimedia Internet Based Languages", to appear in *IC 2004,Procs. 5th Int. Conf. on Internet Computing*, Las Vegas, June 21-24, 2004.

[6] X. Gu, et al, "An XML-based Quality of Service Enabling Language for the Web", *Journal of VLC*, vol.13, no. 1, pp. 61-95, 2002.

[7] D. H. Kim, K. H. Lee An Extended Object Composition Model for Distributed Multimedia Services, Proceedings of the Seventh International Workshop on Object Oriented Real-Time Dependable Systems, January 2002, pp. 279-288.

[8] P. Venkat Rangan, S. Ramanathan, and S. SampathKumar, Feeback Techniques for Continuity and Synchronization in Multimedia Information Retrieval, ACM transactions on Information Systems, Vol. 13, No.2, 1995, pp. 145-176

[9] *Synchronized Multimedia Integration Language 2.0 Specification*, http://www.w3.org/TR/smil20/, Aug. 2001.

[10] Virtual Reality Modeling Language(VRML) 2.0, ISO/IEC 14772,Available:http://www.web3d.org/x3d/specifications/vrml /ISO_IEC_14772-All/index.html

[11] *Extensible Markup Language*-http://www.w3.org/XML