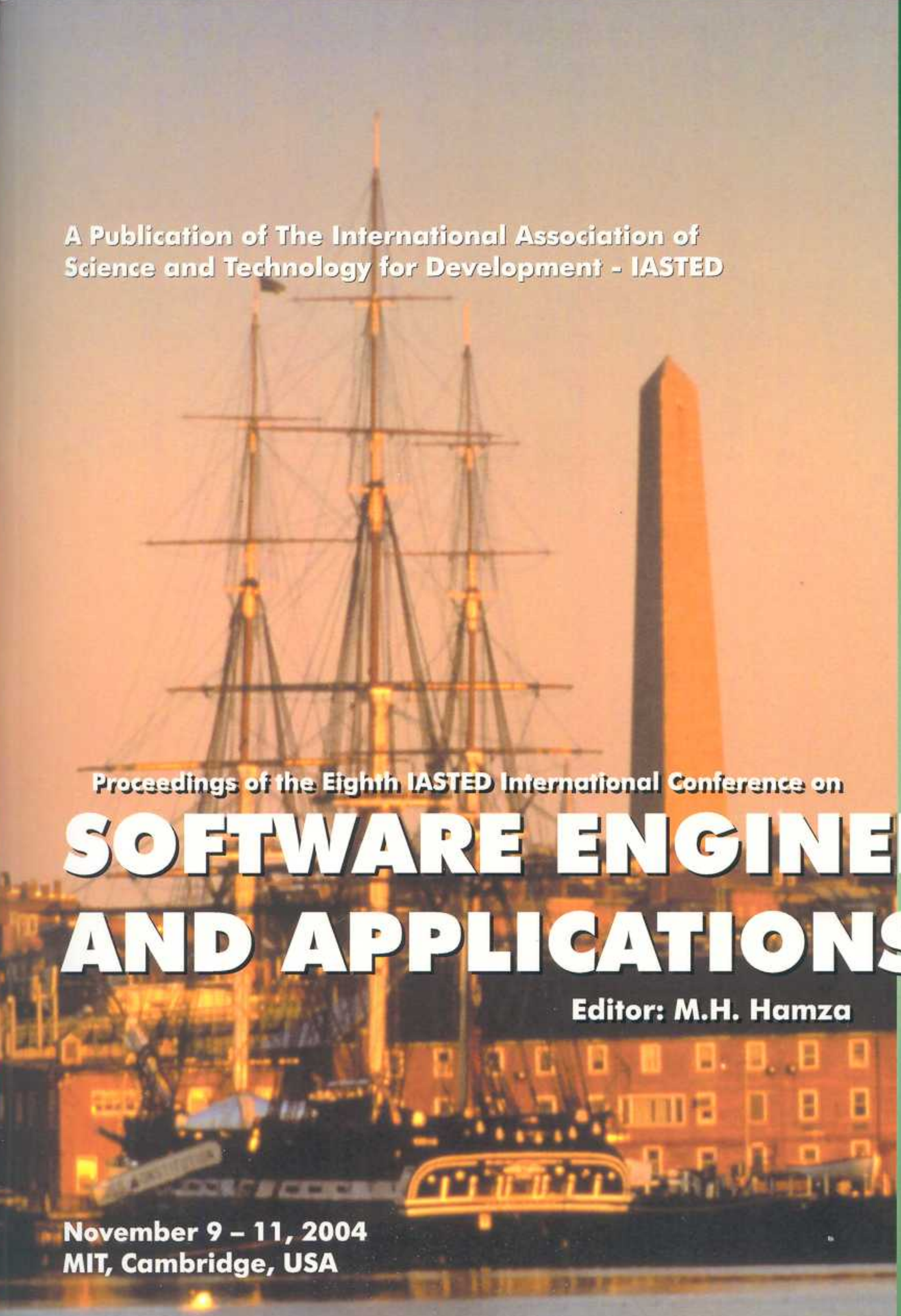

A Publication of The International Association of Science and Technology for Development - IASTED

IASTED

Proceedings of the Eighth IASTED International Conference on

SOFTWARE ENGINEERING AND APPLICATIONS

Editor: M.H. Hamza

November 9 – 11, 2004
MIT, Cambridge, USA

ISBN: 0-88986-425-X
ISSN: 1482-7905

ACTA Press

Anaheim • Calgary • Zurich

**Proceedings of the IASTED International Conference on Software Engineering and Applications (SEA 2004)
Cambridge, MA, USA, 2004**

TOWARDS LARGE SCALE VOICE ACTIVATED DYNAMIC AND INTERACTIVE INTERNET BASED ANIMATION AND MODELING

Angela Guercio, Bonita Simoes, and Arvind K. Bansal
Distributed Multimedia and Cognition Laboratory
Department of Computer Science
Kent State University, Kent, OH 44242, USA
Email: guercioa@hiram.edu, bsimoes@cs.kent.edu and arvind@cs.kent.edu
Phone: +1.330.672.9035, FAX: +1.330.672.7824

ABSTRACT
Multimedia communication over the Internet has gained popularity in recent years. An efficient means of communication is vital for multimedia movies to reach consumers who can use voice commands to interactively modify XML based animations and movies depending on their subjective need. However, the initial modeling of an XML based multimedia movie is low level, time consuming, and not suitable for large scale multimedia animation and modeling. In this paper, we integrate the concept of dynamically modifiable XML scripts and the TANDEM language — a high level XML based Internet multimedia modeling language to transmit asynchronous, nondeterministic, and deterministic events to model and communicate web based multimedia streams over the Internet — to model dynamically modifiable Internet based interactive multimedia animated movies. The integration has been illustrated using a realistic example.

KEY WORDS
Dynamic XML, Internet computing, modeling languages, multimedia communications, voice interaction, web movies.

1. Introduction

With the availability of high-speed Internet, the pervasiveness of PDA's as multimedia communicating devices, and the availability of XML [1] and MPEG-4 [2], the demand for multimedia communication is exploding. People to people multimedia communication and web based collaboration is increasing in addition to multimedia news clips, web based multimedia instructions, and the availability of web based archives of multimedia clips, games, children stories, and movies. In the near future, the availability of ubiquitous multimedia communication will be used to model scenarios involving multimedia movies, virtual animation of 3D realistic models of human and animate characters, and also modify them subjectively according to a client's need. An off-the-shelf library of such movies will be created which can be altered easily to meet a customer's requirements based on voice activated modification by the customer.

These demands require the development of a high level multimedia programming paradigm that will provide user friendly integration of 3D virtual characters, Internet based media stream manipulation, computability, event based modeling with a notion of *loose temporal constraints*, dynamic grouping and synchronization of media streams and 3D virtual objects as well as the flexibility of modeling real world phenomena.

In this paper, we describe our effort to develop such a paradigm. Our paradigm integrates our recent efforts to develop voice activated dynamic modification of Internet based movies [3] and the development of a new high level Internet based multimedia modeling language TANDEM [4, 5] that can model real world phenomena involving tandem events with a notion of *loose temporal constraints*, asynchronous user interactions, transformation and synchronization of Internet based media streams, and computability. We show that manipulation of the media stream can be performed through the use of the transformer, and animation can be simulated and rendered both through the use of the original predefined script commands as well as a set of commands given from live voice. The movie is altered dynamically both during rendering and in the original file description, by dynamically generating XML code.

The TANDEM language is in the advanced stage of implementation, and the voice activated XML script modification system has been implemented.

The major contributions of this paper are:

i. Integration of voice based interactive modification of multimedia movies, dynamic modification of XML scripts, Internet based multimedia stream transformation, and computability.
ii. Dynamically modifiable XML scripts have been introduced as a paradigm to describe interactive modifiable multimedia movies.
iii. TANDEM language has been applied to model the dynamically modifiable XML based movies.

The paper has been organized as follows: Section 2 describes the background related to multimedia modeling, multimedia movies, and multimedia streams. Section 3 describes the TANDEM execution model. Section 4 describes the relevant TANDEM language constructs. Section 5 describes the modeling of XML based movies. Section 6 describes the voice based modification of multimedia movies involving 3D virtual characters and script modification. Section 7 gives an example application of a TANDEM system. Related work is discussed in Section 8 and Section 9 concludes the work.

2. Background

A *multimedia stream S* is defined having two components: *attribute-set* and *data*. Each attribute is a nested tuple. The attribute set varies according to the different media types. An animated movie is a multimedia stream composed of meaningful sequence of scenes. Each scene is a sequence of frames derived using dynamic composition of computationally generated animated objects (possibly mixed with other multimedia streams such as audio and text) that can be grouped together to maintain a common attribute such as synchronization, frame rate etc. or selected programmatically based upon an event or computation. Each frame represents a group of spatially constrained multimedia objects at a particular instance of time.

Each *multimedia object* is composed of a hierarchical graph based representation referred to as the *object graph* [3]. A 2D object has the same graph based representation as 3D objects except that 2D objects are represented as spatially constrained sets of pixels, and 3D objects are represented as spatially constrained sets of meshes.

Each node or edge in the object graph represents a part of an object. Each edge represents the relationship between the two nodes involved. A sub-graph is embedded inside a node, and could represent another complex object. Each sub-graph has a center of gravity attached to it. The *center of gravity* of child nodes is relative to the center of gravity of the parent node, and follows the motion and transformation of the parent node.

Streams coming from a source in a multimedia system can be *periodic* or *aperiodic*. The pressing of a button or a voice command are examples of aperiodic signals. Users interact with the system by generating aperiodic signals, which may cause reactive events to start. An *event* is a user defined temporally constrained set of conditions, which starts a trigger when the conditions are satisfied. A *trigger* reacts by invoking an appropriate action. For example, an event can preempt or abort rendering of a media stream, start another event, start a computation, or transform attributes of a stream or a group of streams. *Grouped streams* work in lock-step fashion and reflect the synchronization constraints on objects of the same group. A *sync point* is an explicit annotation (user defined or automatically generated) to rendezvous two or more media streams to maintain synchronization.

3. TANDEM Execution Model

TANDEM (<u>T</u>ransmitting <u>A</u>synchronous <u>N</u>ondeterministic and <u>D</u>eterministic <u>E</u>vents in <u>M</u>ultimedia <u>S</u>ystems) is an XML based high level language used to model reactive distributed multimedia systems [5]. It supports content-based analysis of multimedia streams, hierarchical grouping of multimedia streams, loose synchronization of complex events, and asynchronous signals from the external environment (which may cause preemption of multimedia streams), dynamic transformation of multimedia attributes, and calls to computational units. TANDEM applications are based on the conceptual model depicted in Figure 1. In Figure 1, the * next to the entity name indicates multiple occurrences of the entity.

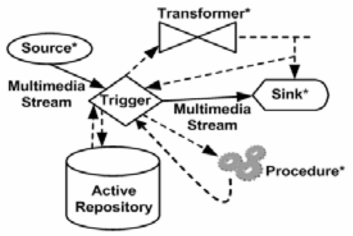

Figure 1. TANDEM execution model

Sources and sinks are respectively origins and destinations of media streams. The reactivity (satisfying real-time constraints) and the interactivity (allowing user interaction) are supported by the *trigger*, which is associated with sets of input multimedia streams, a set of partial conditions (the constraints to satisfy), and an *active repository* (the data processing and storage unit).

An active repository [5] is a persistent database that is responsible for archiving user defined conditions, read-only content-based analysis of the multimedia streams, and verifying the occurrence of conditions in multimedia streams based on content based analysis. The active repository generates one or more internal signals to the trigger in response to the satisfaction of pre-defined conditions describing an event. The persistent archiving and retrieval of past analysis allows for loose ordering of events (tandem events) to occur [5].

The events generated by the trigger are directed either to other triggers for cascading chain of events, or to procedures for generic computation, or to transformers for attribute modification of the multimedia streams. Transformers perform computations that change the attributes of input streams. For example, transformers are used to change the rendering rate of a group of streams, to multiplex streams, to reduce the number of channels of a video for rendering purposes or to change the motion of animated components. A *procedure* performs generic computation. For example, procedures are used to compute the traffic over the Internet, to perform voice interpretation, or to sort objects. The output of a procedure is returned to the trigger, the output of the transformer is transmitted to either the trigger or another transformer or a *sink* that archives the multimedia stream for later use or renders the multimedia stream.

4. TANDEM Language Constructs

In this section we give a quick overview of the TANDEM constructs and we refer the reader to [4, 5] for more details. A TANDEM *application* contains many streams and multiple triggers. Each trigger performs operations on one or many groups of input multimedia streams. Periodic multimedia streams can be *audio*, *video*, *text, animation* of 3D objects, and so on. *Aperiodic* signals are also part of a trigger but not part of groups.

A media stream provides the source name, its URI, the media type as well as the specific attribute constraints that the media must satisfy. For example, the following TANDEM excerpt describes an audio stream, transmitted at 44,100 samples/sec from a remote source.

```
<media_stream name = "mm1">
  <source name = "source1" URI = "192.168.2.102" />
    <type> <audio name = "audio.wav"
        samples_per_sec="44100"  /> </type> </media_stream>
```

Grouping clusters one or more media streams or groups for synchronization. This excerpt groups two elements.

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

The groups are both *dynamic* and *hierarchical*. Elements are dynamically ungrouped in the transformer. This transformer ungroups "my_group".

```
<transformer name = "ungroup_transformer">
  <action> <ungroup>
       <elements group = "my_group"/></ungroup> </action>
  <dest name = "destination_name" /> </transformer>
```

The *action* in the transformer contains constructs to set or modify attribute values of the media streams. Script languages can be used to support such actions.

Events are generated by the trigger when associated conditions are satisfied. Conditions are checked by the active repository. All the conditions need not be satisfied at the same instant. This supports loose ordering of events. Events can start immediately when conditions are satisfied, unless delay states otherwise. The syntax of an event with default values follows.

```
 <event name="EventName"  start="0" priority="1"…>
   <partial_condition name="cond1">…</partial_condition> . . .
   <partial_condition name="condn"> …</partial_condition>
   <destination name = "DestinationName"  … />  </event>
```

TANDEM supports nested loops, concurrency, and preemption [4, 5]. Loops are used to execute a stream multiple times. Abort commands can be inserted in a loop when preemption or suspension of the stream(s) is required. In the following excerpt, rendering of the stream is suspended when *Button1* is pressed and restarted when the *ReleaseButton1* is pressed.

```
<loop times = "1">
   <abort when = "Button1" type="strong"
          suspend="true" resume="ReleaseButton1"/>
    <loop_element name = "mm1" /> </loop>
```

Abortion modeling has been inspired by [6]. However, it is quite different due to the delay caused by multimedia rendering until sync points. There are two types of abortion: *strong and weak*. *Strong abortion* aborts the loop right after the multimedia sync point while *weak abortion* aborts after the current clip has been rendered. Detailed TANDEM constructs and semantics are omitted due to space limitation.

5. Modeling 3D Object Based Movies in XML

An XML movie has attributes such as the number of frames per second, the frame size, and the position of a display window. An excerpt of a movie is illustrated in Figure 2. Each frame is associated with a time instance and is a collection of media objects modeled as hierarchical graphs [3]. Each object has a name, and the corresponding object graph representation. An object could be 3D mesh-based, a 2D image, audio or audiovisual data. Images are used to describe inert objects in a scene such as a wall hanging or a portrait. Audio can be used to add sound effects to a scene such as background music and sound effects. Audio can be incorporated in the movie or streamed separately. Text can be used for captions and titles.

Background is modeled as a collection of one or more images or media streams. Surroundings consist of 3D immovable objects modeled as meshes (i.e. a room), lighting effects, and sound effects. The animation of media objects is achieved using transformation matrices associated with one or more groups of complex objects. During rendering, the scene graph is traversed starting from the root node, so that the appropriate transformation, clipping and lighting can be applied to the objects. An action represents a complex sequence of animations or a sequence of low level actions. An action could be repetitive such as walking (a repetitive loop of steps).

The camera is represented by the scene perspective, field of view, and by its position and rotation [2]. The animation speed describes the rate of rendering, and movement speed describes how fast the actual motion of the object occurs in relation to other motion. We refer the reader to [3] for more details.

```
<movie fps="30" width="640" height="480" xpos="0" ypos="10"
      style="window" num_scenes="1" caption="My story">
   <scene id="1" name= "Scene1" num_objects="5"
        texfname="Scene" music="song.mid" height="32.0">
    <background background_id="1">
        <background_type>matrix</background_type>
        <texture_type>jpg</texture_type>
        <num_images>6</num_images>
        <rows>2</rows><columns>3</columns></background>
    <resources>…</resources>
    <perspective fov="0.6021124" aspect="1.3333"
        near="10.0" far="20000.0" />       …
    <scene-mesh scene-name="SceneMesh1">
        <scene-vertices  total="229">…</scene-vertices>
        <scene-faces total="325" sides="3">…</scene-faces>
        <scene-materials num-materials="2"> …
             <features> … </features>…
        </scene-materials> </scene-mesh>
    <objects> <object3D object-id="0" object-name="Tiny">
        <graph graph-id="0" node-count="47">…</graph>
        <object-mesh object-meshname="Tiny" meshnum="0"
             attach_to_graph="0" attach_to_node="Body"> …
        </object-mesh>
        <sets-of-animations>
            <animation-set setname="Walk" looping="true"
               animationspeed="4" movementspeed="0.15">
```

```
                <node-properties>
                  <graph-node node-name="Root"
                      numinstances="62" key-type="matrix" />…
                 </node-properties>
                 <time-instance value="0">
                   <node node-name="Root" transform="…"/> …
                 </time-instance>
                 <time-instance value="80">
                   <node node-name="Root" transform="…"/>…
                 </time-instance>
               </animation-set>
           </sets-of-animations>
           <object-controller>
             <default-anim-speed>6</default-anim-speed>
             <default-motion-speed>0.15</default-motion-speed>
             <default-animation>Walk</default-animation>
             <move x="-100.0" y="0.0" z="10.0" />
             <scale x="0.25" y="0.25" z="0.25" />
             <rotate x="-1.57" y="1.57" z="1.57" />
          </object-controller> </object3D>
          <object3D objectid="1" objname="Zombie">
          …</object3D>…<voicecommands>……<script>…
```

Figure 2. A fragment of "movie.xml"

6. Voice Commands and Dynamic Scripts

An XML movie has a predefined script associated with it. The system has the capability of disambiguating between multiple objects and actions, and supports aliasing for user-friendliness. An excerpt of the *voicecommands* tag and the *script* tag for a movie is shown in Figure 3. In the rule name *person* the names "Zombie" and "Monster" represent the same object. Similarly, synonymous actions "move" and "go" lead to the same set of low level animations.

The second *auxiliary* rule contains a list of auxiliary commands used to help the user with terminating the movie player or with help about the structure of the voice commands. An animation that loops such as walking will have an amount to specify the number of steps the object should walk. This implementation is designed to have looping animation make use of the amount rule whereas non-looping animations do not.

The script defined in a movie plays initially in the sink. However, the system can be interrupted, and the movie can be archived by user interaction using a menu selection 'Start Recording'. From that point on, the rest of the script is discarded and the voice commands from the microphone form the basis of interaction. Each voice command that is recognized by the speech engine and the low level script corresponding to the action is recorded in memory. The user can save the modified script after interacting with the objects in the movie.

Assume that the user speaks the following commands into the microphone after the second command in Figure 3 was executed: (i) "Tiny walk right five", (ii) "Zombie move left", (iii) "Zombie jump", and (iv) "Tiny walk left one". The third command in the above list will be discarded as an invalid command since jump is not recognized as a valid action. After command (iv) above is executed, the user saves the XML file from the menu selection either to a new file or overwrites the existing XML movie file. The resulting *script* tag will be changed to reflect the interaction as shown in Figure 4.

```
<voicecommands>
  <RULE ID="1" NAME="commands" TOPLEVEL="ACTIVE">
    <RULEREF NAME="person" />
    <RULEREF NAME ="action" />
    <RULEREF NAME ="direction" />
    <o><RULEREF NAME ="amount" /> </o> </RULE>
  <RULE ID="2" NAME="auxiliary" TOPLEVEL="ACTIVE">
    <LIST>
      <PHRASE propname ="actions"> actions</PHRASE>
      <PHRASE propname ="actions">commands</PHRASE>
      <PHRASE propname ="close">close</PHRASE>
    </LIST> </RULE>
  <RULE NAME="person">
    <LIST><PHRASE propname ="Tiny ">Tiny</PHRASE>
         <PHRASE propname ="Zombie">Zombie</PHRASE>
         <PHRASE propname ="Zombie ">Monster</PHRASE>
    </LIST> </RULE>
  <RULE NAME="action">
     <LIST><PHRASE propname ="Walk ">move</PHRASE>
             <PHRASE propname ="Walk ">go</PHRASE>
             <PHRASE propname="Idle ">stand idle</PHRASE>
    </LIST> </RULE>
  <RULE NAME="direction">
    <LIST><PHRASE propname ="right ">right</PHRASE>
            <PHRASE propname ="left ">left</PHRASE>
            <PHRASE propname ="up ">up</PHRASE>
            <PHRASE propname ="down ">down</PHRASE>
    </LIST> </RULE>
  <RULE NAME ="amount">
    <LIST>
      <PHRASE propname="1">one</PHRASE>
       …
      <PHRASE propname="20">twenty</PHRASE>
    </LIST> </RULE> /voicecommands>
<script>   <command>Tiny walk right one</command>
           <command>Zombie walk hurt left two</command>
           <command>Zombie move left one</command>
</script>
```

Figure 3. Voice commands and script

```
<script>
          <command>Tiny walk right one</command>
          <command>Zombie walk hurt left two</command>
          <command>Tiny walk right five</command>
          <command>Zombie move left</command>
          <command>Tiny walk left one</command>
</script>
```

Figure 4. A newly generated script

The first two commands are related to the original script in the XML file. The third command in Figure 3 is discarded since the user interrupted the movie at that point. The last three commands in Figure 4 reflect the user's interaction that is appended to the old script. This makes up the new script used for rendering and archiving. Only *animation related* commands are written to the script. The *auxiliary* commands used to help the user are not recorded in the script.

Figure 5 shows the algorithm that replaces the old text script in the original XML movie with a new script containing the user's voice command interaction. The XML document is searched to retrieve a scene node. A new *XmlElement* "script" node is then created. The script played from the original XML file and the user's voice command interactions are then appended to the new script element. This element is then written, using the

XmlTextWriter class, into the new XML file denoted by the variable *newFile*.

```
Algorithm: ReplaceScript
Input: XML Movie file
Output: A new XML Movie file, newFile
{XmlNode* root = myXmlDoc->DocumentElement;
 XmlNode* node; node = root->FirstChild;  //get the root node
 while (node != 0) {
  if ( a scene node is found ) {
      XmlElement* elem = myXmlDoc->CreateElement("script");
      for (every script command that was executed)
          elem->AppendChild(movie script command);
      for (every valid voice command issued)
          elem->AppendChild(voice command);
      // replace old script node
      node->ReplaceChild(elem, node->LastChild);} // end if
  // keep iterating if scene is not found
  node = node->NextSibling;} // end while
 XmlTextWriter* writer; writer = new XmlTextWriter(newFile,0);
 // preserve XML indenting
 writer->Formatting = Formatting::Indented;
 // write the replaced node to the new XML file
 myXmlDocument->WriteTo(writer);} //end ReplaceScript
```

Figure 5. An algorithm to replace a script node

7. TANDEM Based Dynamic Movie Scripts

In this section we show how the TANDEM model is used to describe dynamic voice modifiable XML movies. Original movies are formed by synthesized animation and voice input that controls the outcome of the story in the movie. The TANDEM system analyzes the movie, detects the components of scenes, modifies the attributes of the movie, and renders or archives the modified movie. A simplified version of the TANDEM application is given in Figure 6.

The rendering of the movie starts immediately as soon as the conditions of the *invoke_compute0* event are satisfied (i.e. when *movie.xml* is available). The rendering of *Scene1* will be performed 3 times in a row as required by the loop construct. The rendering will be interrupted when the *Start Recording* signal is detected, as the abort in the loop states. The processing of the movie requires the parsing and the interpretation of the animation commands in the file. These are performed in the procedure module that is activated, respectively, once by the *invoke_compute0* and a second time by the *invoke_compute1* event. The second time, the *interpret_script* procedure produces data required for setting attributes (such as component's name, action to perform, direction of action and duration of action). The attributes are set by the *MotionTransformer* which is activated by the *invoke_compute2* event.

If live voice interaction is desired, the *Start Recording* button is pressed. Voice is an animation related aperiodic stream that instructs the trigger to activate a transformer which in turn modifies the movie accordingly. The processing now uses a voice sub-system [3] that consists of a speech recognition module, and a voice command interpreter. The speech recognition of the stream is performed in the *active repository*, which performs the content based analysis of the voice stream, as stated by the *invoke_compute3* event. Finally, the *interpret_script* procedure can be initiated and the sequence proceeds as in the previous case.

If the *Save* button is pressed then the *invoke_compute4* event is generated and the *MotionTransformerandSave* transformer is activated. The transformer sets the attributes and has as destination the *DynamicScripting* transformer which dynamically changes the scripts and archives the new movie in a new file.

At the beginning of the application, two components of the movie are grouped together in the trigger so that the action identified on one component will be repeated simultaneously by the other component. This is how grouping is realized.

```
<application>
<media_stream name="my_movie" >
   <source name="source1" URI="192.42.31.2"/> <type>
       <movie_animation name="movie.xml"/> </type> </media_stream>
<trigger name="trigger1">
   <loop name="Scene1_Loop"  times="3">
     <abort when="Start Recording" type="strong" />
     <loop_element name="Scene1" /> </loop>
# grouping causes Zombie and Tiny to act simultaneously
  <group name="Zombie&Tiny">
   <member name="Zombie"/> <member name="Tiny"/></group>
# triggers the movie script parsing
   <event name="invoke_compute0" start="0">
       <partial_condition name="cond1" signal_type="movie.xml"
           presence="present"/>
       # test the possibility to start a procedure
       <partial_condition name="cond2"
         signal_type="procedure_start" presence="present"/>
       <dest name ="parse_script"/> </event>
# triggers the movie script interpretation
   <event name="invoke_compute1" start="0">
       <partial_condition name="cond1" signal_type="movie.xml"
           presence="present"/>
       # test the possibility to start a procedure
       <partial_condition name="cond2"
         signal_type="procedure_start" presence="present"/>
       <partial_condition name="cond3"
          signal_type="parse_script" presence = "present"/>
       <dest name ="interpret_script"/> </event>
# triggers the transformation that will produce the animation
   <event name="invoke_compute2" start="0">
       <partial_condition name="cond1" signal_type="movie.xml"
           presence="present"/>
       <partial_condition name="cond2"
         signal_type="interpret_script" presence = "present"/>
       <partial_condition name="cond3" signal_type="save_button"
             presence="absent"/>
       <dest name ="MotionTransformer"/> </event>
# triggers the speech recognition in presence of incoming voice
   <event name="invoke_compute3" start="0">
       <partial_condition name="cond1" signal_type="movie.xml"
             presence="present"/>
       <partial_condition name="cond2"
         signal_type="StartRecording" presence="present"/>
       <partial_condition name="cond3" signal_type="my_voice"
             presence  =  "present"/>
       <partial_condition name="cond4">
             # speech recognition is applied in the active repository to the
             incoming voice
             <match signal_name="my_voice" template="WordList"
            method="speech_recognition" threshold="90%"/>
       </partial_condition>
       <dest name = "interpret_script" /> </event>
# start the transformer for the animation and save the result
   <event name="invoke_compute4" start="0">
       <partial_condition name="cond1" signal_type="movie.xml"
             presence="present"/>
       <partial_condition name="cond2" signal_type="interpret_script"
             presence  = "present"/>
       <partial_condition name="cond3" signal_type="save_button"
             presence="present"/>
       <dest name="MotionTransformerAndSave" /> </event>
# parse movie scripts
```

```
    <procedure name="parse_script" start="0">
        …#parse the scripts…</procedure>
# interpret scripts
    <procedure name="interpret_script" start="0">
        …#interpret the scripts…</procedure>
# set the attributes before sending it to be rendered
    <transformer name =  "MotionTransformer”>
        <action><update_attributes>
       …#code to set the attributes for the movie.
                </update_attibutes> </action>
        <dest name  =  "XMLMoviePlayer" /> </transformer> </trigger>
# set the attributes before sending it to be rendered
    <transformer name="MotionTransformerAndSave">
        <action><update_attributes>
                …#code to set the attributes for the movie.
                </update_attibutes></action>
        <dest name="DynamicScripting" />
        <dest name=“ XMLMoviePlayer” /> </transformer>
#replace the old scripts in movie.xml with the new ones when voice
commands are present
    <transformer name="DynamicScripting">
        <action><save_script>
          …#dynamically update movie.xml  </save_script></action>
        <dest name = “newfile.xml” /> </transformer>
</application>
```

Figure 6. A TANDEM modeling application

8. Related Work

We have been influenced by many contemporary research areas [6, 7, 8, 9, 10, 11] in recent years which have focused on graphics animation [11], speech recognition systems [10], WWW consortium research on modeling scenes and objects in MPEG-4 [1] and the use of XML in MPEG-7 [12]. There has also been research on avatar based virtual objects [13], event cascading in VRML [8] and their XML based variants [14, 15]. The concept of synchronization and event [16] based preemption is found in ESTREL [6]. We also built on our own research on transmitting multimedia using graph based modeling of complex objects [3]. However, our notion of the use of integration of asynchronous signals, an active repository to model tandem events with loose temporal constraints as well as rigid temporal constraints, the integration of computability, triggers, media attribute transformations, dynamic selective grouping of media streams to meet media attribute constraints such as synchronization and dynamic modification of XML scripts are not present in these related works to a varying extent.

9. Conclusion and Future Work

In this paper, we have described a paradigm for the high level description of dynamically modifiable interactive Internet based multimedia movies. The paradigm is based on the integration of embedding 3D graph based objects and media object commands into XML scripts and real time voice activated dynamic modification of XML scripts. It is also based on the use of high level media stream grouping, dynamic transformation of media attributes, and the use of an active repository to model tandem events with loose temporal constraints. The integration has been achieved using the high level language TANDEM and a voice activated XML based system that modifies XML based movies. Currently the focus is on the full implementation of various constructs of the TANDEM language so we can develop new exciting applications using this paradigm.

References:

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

[2] H. Kalva, L. Cheok, A. Eleftheriadis, MPEG-4 systems and applications, *Proc. 7th ACM Intl. conf. on Multimedia (Part 2)*, Orlando, Florida, October 1999, 192-192.

[3] B. Simoes and Arvind K. Bansal, Interactive Voice Modifiable 3D Dynamic Object Based Movies over the Internet, *Proc. 5th Intl. Conf. on Internet Computing*, Las Vegas, June 2004, 708-714.

[4] A. Guercio, A. K. Bansal, A Model for Integrating Deterministic and Asynchronous Events in Reactive Multimedia Internet Based Languages, *Proc. 5th Intl. Conf. on Internet Computing*, Las Vegas, June 2004, 46-52.

[5] A. Guercio, A. K. Bansal, TANDEM – Transmitting Asynchronous Non Deterministic and Deterministic Events in Multimedia Systems over the Internet", *Proc. 10th Intl. Conf. on Distributed Multimedia Systems,* San Francisco, September 2004, to appear

[6] G. Berry, G. Gonthier, The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation, *Science of Computer Programming*, vol. 19, no. 2, Nov. 1992, 87-152.

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

[8] Virtual Reality Modeling Language (VRML) 2.0, ISO/IEC14772,http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/part1/concepts.html,Date accessed: 07/01/2004.

[9] K. Perlin and A. Goldberg, Improv: A System for Scripting Interactive Actors in Virtual Worlds, *Proc. 23rd annual Conf. on Computer graphics and interactive techniques*, 1996, 205-216.

[10] H. Tanaka, T. Tokunaga, Y. Shinyama, Animated Agents that Understand Natural Language and Perform Actions, *Proc. International Workshop on Lifelike Animated Agents, (LAA),* Tokyo, Japan, 2002, 89-94.

[11] eXtensible 3D(X3D), http://www.web3d.org/x3d/overview.html, Date accessed: 07/01/2004

[12] J.M. Martinez, R. Koenen, F. Pereira, MPEG-7 – The Generic Multimedia Content Description Standard, Part I” *IEEE Multimedia, April-June Issue*, 2002, 78-87.

[13] S. Kshirsagar, N. Thalmann, A. Vuillème, D. Thalmann, K. Kamyab and E. Mamdani, Avatar Markup Language, *Proc. Workshop on Virtual environments,* Barcelona, Spain, 2002, 169-177.

[14] Z. Huang, A. Eliens and C. Visser, XSTEP: An XML-based Markup Language for Embodied Agents, *Proc. 16th International Conf. on Computer Animation and Social Agents, (CASA 2003),* New Brunswick, New Jersey, 2003, 105-110.

[15] K. Walczak and W. Cellary, X-VRML – XML Based Modeling of Virtual Reality, *Proc. IEEE Symposium on Applications and the Internet,* SAINT’02, Nara City, Nara, Japan, 2002, 204-213.

[16] X. Gu, K. Nahrstedt, An Event-Driven, User-Centric, QoS-aware Middleware Framework for Ubiquitous Multimedia Applications, *Proc.ACM Multimedia Workshop on Middleware*, Ottawa, Canada Oct. 2001, 64-67