Eigth Eurographics Workshop on Virtual Environments (2002)
S. Müller, W. Stürzlinger (Editors)

Avatar Markup Language

Sumedha Kshirsagar
Nadia Magnenat-Thalmann
MIRALab, CUI 24 rue du General Dufour
CH 1211 Geneva, Switzerland

Anthony Guye-Vuillème
Daniel Thalmann
LIG, EPFL
1015 Lausanne, Switzerland

Kaveh Kamyab
Ebrahim Mamdani
IIS, Imperial College
London SW7 2BT, UK

Abstract

Synchronization of speech, facial expressions and body gestures is one of the most critical problems in realistic avatar animation in virtual environments. In this paper, we address this problem by proposing a new high-level animation language to describe avatar animation. The Avatar Markup Language (AML), based on XML, encapsulates the Text to Speech, Facial Animation and Body Animation in a unified manner with appropriate synchronization. We use low-level animation parameters, defined by the MPEG-4 standard, to demonstrate the use of the AML. However, the AML itself is independent of any low-level parameters as such. AML can be effectively used by intelligent software agents to control their 3D graphical representations in the virtual environments. With the help of the associated tools, AML also facilitates to create and share 3D avatar animations quickly and easily. We also discuss how the language has been developed and used within the SoNG project framework. The tools developed to use AML in a real-time animation system incorporating intelligent agents and 3D avatars are also discussed subsequently.

Keywords:

Avatar animation, Virtual human animation, Animation language, Agent controlled animation.

1. Introduction

Avatar animation is no longer a mere research topic. Many efforts in this area, combined with standardization (*e.g.* MPEG-4 facial and body animation), have lead to a wide use of virtual faces and bodies for the interactive web based and entertainment applications. Recently, very believable virtual humans have been seen in the movies. However, such production demands many days or even months of manual design work from highly skilled animators. On the contrary, real-time avatar animation on the web cannot be completely pre-defined with the accuracy and the artistic precision evident in the computer generated movies of today. The advent of web-based e-commerce applications demand powerful and easy to use high-level interfaces, capable of animating web-based characters, instantly and smoothly. Such an interface finds immediate applications for web-based virtual sales assistants, controlled by an intelligent agents, and face-to-face avatar interaction. In recent years, there have been several efforts for the development of such a high-level control mechanism for character animation, from IMPROV [1] to BEAT [2]. The IMPROV system suggested use of high-level scripts and tools for creation of real-time behavior based animation with non-repetitive motions and smoothness. The system allowed animators to create rules governing how the synthetic actors communicate and make decisions. More recently, BEAT (Behavior Expression Animation toolkit) used text input to create behavioral animation, using the behavioral rules defined by the animator. The toolkit generated appropriate and synchronized nonverbal behaviors and synthesized speech. In spite of all such efforts, the current shared virtual environment systems still rely on limited iconic interactions and pre-defined gestures like “smile” and “wave”. For example, several applications have been developed on the Microsoft Virtual World platform [3] and the Blaxxun community platform [4]. However, it is evident that there are several limitations on the interactivity of today’s integrated systems, in spite of use of 3D face and body representations. It appears that it is difficult for a common web user to quickly and easily design animations for her own avatar. One of the solutions to reduce this difficulty, can be to drive the avatar by an intelligent agent that accepts natural language commands. In this case, we need an interface between the intelligent agent and the avatar animation engine.

This paper aims to provide such an interface by specifying a new animation language. The Avatar Markup Language (AML) enables high-level

specification of avatar animations, to be used by intelligent agent systems as well as web users and designers, with ease and rapidity. We choose MPEG-4 Facial Animation Parameters (FAP) and Body Animation Parameters (BAP) as the basis for the animation [5,6]. However AML is not constrained by the use of MPEG-4 animation parameters. An advantage of AML is that it provides a powerful high-level animation interface, without requiring intricate knowledge of the low-level animation parameters (MPEG-4 or otherwise) controlling the 3D avatar. The discussion in the paper is limited to the development and definition of AML. The use of AML in an animation system can exploit various possible techniques for face and body animations, for example[7,8], and hence not elaborated here.

We refer here to another prominent effort to solve the problem of interaction with the virtual characters using only high level language, VHML (Virtual Human Markup Language) [9]. This proposal under development uses various sub-elements to control speech, face and body animations of a virtual human. A variety of tags are defined to control various parameters relating to virtual human animation in a structured manner. Though it is an independent development, VHML is an XML based scripting language with similar aims to AML. We delay the discussion on VHML in the light of comparison with AML till Section 5. We begin by describing the requirements and needs that led to the development of AML, within the framework of the project SoNG. Section 3 defines the syntax and various elements of AML and how it can be used in defining unified face and body animations along with Text to Speech (TTS) capability. Section 4 presents a detailed explanation of a complete scheme using AML in an MPEG-4 compliant animation system. We conclude with discussion on other possible applications of AML and future directions for development.

2. The SONG Project

SoNG (portalS of Next Generation) is a European Union funded project that commenced in January 2000 as a consortium of 13 European academic and industrial partners. Portal is a term synonymous with an access point to resources and services on the Web. Typical services offered by portals include: directory of resources, search facilities, news, e-mail, voice chat, map information, and sometimes community forums. These services generally rely on point-and-click on structured information like text, still pictures or 2D graphics. The project intends to investigate, develop and standardize the building blocks for the next generation of portals. These building blocks include existing Web technologies, but also 3D computer graphics elements as in computer games, intelligent agents embodied in realistic avatars, new user-friendly interfaces and real-time audio-visual communications. The technology integration platform is aimed to be a fully MPEG-4 compliant player. The project will demonstrate how these new technologies when integrated into a sample e-commerce application allow an easier and more natural access to resources and services. In this section we explain how the specifications of such an application led to the development of AML. We would like to emphasize that, though the requirements of this particular project have lead to the development of the AML, maximum efforts have been devoted to make the language independent of specific platform and available animation methods. The requirements discussed here form the important aspects of any interactive system with high level control of virtual humans and intelligently controlled avatars.

2.1. Requirements and Purpose

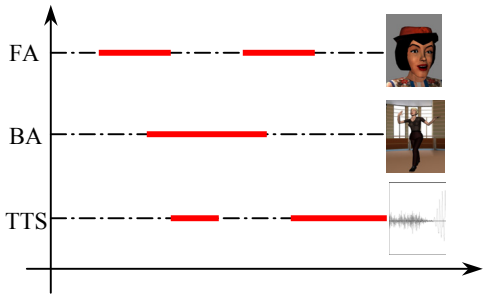

Figure 1. Animation Scenario

To demonstrate the technology developed during the project, the SoNG consortium is building a 3D e-commerce application equipped with a virtual shop assistant controlled by intelligent agents. For this, it was found necessary that the intelligent agents control their 3D avatars, *i.e.* faces and bodies that are fully MPEG-4 compliant, effectively. Yet, the intelligent agents cannot (and need not) communicate to the avatar directly using the low-level MPEG-4 streams. The primary mode of such communication should be mere high-level language comprising of text, expressions and gestures. For example, when a shop assistant welcomes a virtual customer, it would smile, bow a little, and say, "May I help you?". In order to facilitate such multi-modality in its interaction, the agent is required to trigger the appropriate face animation, body animation and TTS modules in a time-synchronized and easy manner. This may involve mixing of multiple gestures and expressions into a single animation, as shown in Figure 1. At the same time, the context in which the agent must operate is often unpredictable and can be a function of the user's actions, events in the environment and/or the agent's own personality and emotions. It may be also necessary to trigger parametric behaviors such as pointing at an object or moving to a specific coordinate in the 3D space. Similarly, it is required to have a high-level interface tool that will

enable end users to control their own avatars in the virtual shop with a similar variety of animations. To this end, it is merely not adequate to use a set of pre-defined facial expressions and body gestures in order to attain believability. Thus, it is necessary to define a high-level language to assemble the interaction modalities into a unified animation while providing the required flexibility. We propose to use AML as the high-level language. We use the MPEG-4 FAPs and BAPs as low-level animation parameters to realize the AML based system. A database of facial expressions and body gestures using MPEG-4 FAPs and BAPs respectively is developed to be used for this purpose. Figure 2 shows selected snapshots of the facial expressions and body gestures that form part of the database used as the building blocks of the avatar animation.

Figure 2. Face and Body Animation snapshots

For the scenario under consideration, Figure 1 shows three tracks, one each for face animation, body animation and TTS. A red line indicates the presence of the particular track during that time interval. Note that the face animation and TTS overlap at times. For example, the avatar may be smiling and starts speaking as it continues to smile. Appropriate co-articulation needs to be implemented while rendering realistic speech animation. The carefully weighted sum of facial animation parameters is required in order to avoid jerky animation or artefacts [10]. In order to have proper body animation synchronized either with speech or with facial expressions, appropriate positioning of these tracks is necessary. The complex scenarios with overlapping animation elements, as described above, must be supported in order to allow flexible and seamless animation.

2.2. MPEG-4 Animation Parameters

As the SoNG platform is completely MPEG-4 compliant, the choice of the MPEG-4 FAPs and BAPs for avatar animation is a natural choice. In this section we give a brief overview of the FAPs and BAPs. A complete and detailed description is beyond the scope of this paper, instead, we refer to [5,6].

MPEG-4 defines certain key locations on the face as feature points. The FAPs are defined in terms of the normalized displacements of these feature points from their neutral positions, thus resulting in different expressions. There are 66 low-level and 2 high-level paraeters. Stretch right corner lip, raise left inner eyebrow, puff cheek *etc.* are examples of low-level FAPs. The two high-level FAPs are: *visemes* and *expressions*. Each *viseme* can take one of the 14 pre-defined values corresponding to groups of phonemes. Each *expression* can take one of the 6 values (joy, sadness, anger, disgust, fear, and surprise) Depending on the application, the high-level parameters can be specified in terms of low-level parameters when precision and variety is of prime importance.

The MPEG-4 body animation is defined in terms of 296 Body Animation Parameters, which represent a certain Degree Of Freedom (DOF) of a given body articulation standardized by the H-Anim [11]. The DOF are translations or rotations along an axis. BAP values specify a specific state of human skeleton *e.g.* sacroiliac tilt, left hip twisting, skull base rotation *etc.* The full body animation requires 186 basic BAPs; 110 other BAPs being reserved for extensions like tail animation or to deform some parts of the bodies.

3. AVATAR MARKUP LANGUAGE (AML)

This section explains how high-level animation description can be achieved by using the AML. We explain the syntax and elements of the AML with examples.

3.1. Syntax and Elements

Figure 3 shows the AML syntax with the basic elements. The root tag <AML> marks the beginning and end of the script. It accepts four attributes: *face_id* (the reference id for the 3D face to be animated), *body_id* (the reference id for the 3D body to be animated), *root_path* (the root path for animation files such as expression files, FAPs and BAPs, explained in detail in the next subsections) and *name* (the name of the animation project). <AML> has 2 sub-elements: <FA> or Facial Animation and <BA> or Body Animation.

```
<AML face id="x" body id="y" root path= "p"
name = "name of animation">
  <FA start_time="t1" input_file= "f1">
  <TTS mode = "m" start_time = "t3" out-
put_fap = "f3" output_wav = "f4">
    <Text>TextToBeSpoken<\Text>
  <\TTS>
  <AFML>…<\AFML>
  <\FA>
  <BA start_time = "t2" input_file = "f2">
    <ABML>…<\ABML>
  <\BA>
<\AML>
```

Figure 3. AML Syntax

The <FA> and <BA> tags both accept two attributes: *start_time* (the relative start time of the respective scripts) and *input_file* (the name of Avatar Face Markup Language (AFML) or Avatar Body Markup

Language (ABML) file if they are predefined; *input_file* can also have the value "none"). In the case that *input_file* is "none" the AFML and ABML are defined within the AML script.

The <FA> further has an optional sub-element <TTS> or Text To Speech. The <TTS> tag accepts four attributes: *mode* ("annotated" if the TTS input is annotated with emotional tags, "plain" otherwise), *start_time* (the relative start time of the TTS rendering), *output_fap* (the filename to use when generating FAP output) and *output_wav* (the filename to use when generating audio output). Though we use the fap file as output here, it may be possible to use any other format (viseme or phoneme), provided the parser and further processing are designed accordingly. <TTS> has a sub-element <Text>, which defines the text to send to the TTS engine. The speech animation can be added either in <TTS> tag or directly in the AFML, as will be explained in the following sub-sections.

3.2. Avatar Face Markup Language

```
<AFML>
  <Settings>
    <Fps>FramesPerSecond</Fps>
    <Duration>mm:ss:mmm</Duration>
    <FAPDBPath>"path for expression (.ex)
    files"</FAPDBPath>
    <SpeechPath>"path for speech animation
    (.vis) files"</SpeechPath>
  </Settings>
  <ExpressionsFiles>
    <File>"expression file name" </File>
  </ExpressionsFiles>
  <ExpressionsTrack name="Track name">
    <Expression>
      <StartTime>mm:ss:mmm</StartTime>
      <ExName>"name"</ExName>
      <Envelope>
        <Point>
        <Shape>{log,exp,linear}</Shape>
        <Duration>InSeconds</Duration>
        <Int>NormalizedIntensity</Int>
        </Point>
      </Envelope>
    </Expression>
  </ExpressionsTrack>
  <SpeechTrack name="name_of_track">
        <StartTime>mm:ss:mmm</StartTime>
        <FileName>"viseme or fap file name"
        </FileName>
        <AudioFile>"FileName"</AudioFile>
  </SpeechTrack>
</AFML>
```

Figure 4. AFML Syntax

We use MPEG-4 FAP as low-level parameters for animation. The FAP Database (FAP DB) contains a variety of pre-defined facial expressions, defined in terms of MPEG-4 FAPs (see Section 4). The <Settings> tag contains the information like the frame-rate, length of animation, and the local path of FAP DB as well as pre-defined speech animations, if any.

A number of tracks can be defined in the AFML. Each track can be given a name; *e.g.* emotions, head movements, eye movements *etc.* This separation enables distinct control over various parts of the face. There may be as many <ExpressionsTrack> elements as required. Further, there may be as many <Expression> elements as required in each <ExpressionsTrack>. The expressions may or may not be overlapping. Each <Expression> has a start time, a name, and a time envelope defined. The name, in fact, refers to the static expression file from the FAP DB. We chose this scheme rather than using a distict set of tags to define expressions, so that the animation is not restricted by the tags defined by the language. The designer of the system can add as many expressions as desired to the FAP DB and all of them are available for use in AFML. Each time envelope is defined by as many <Point> elements as required. The shape defines the interpolation from the previous point to the current one, and can take value of logarithmic, exponential or linear. The first default point is with zero intensity at the start time. The intensity is normalized and duration is specified in seconds. The <SpeechTrack> is reserved for the viseme or fap files corresponding to a pre-defined speech animation. The viseme file contains timing information for each of the viseme and the fap file contains frame-by-frame information of the low-level facial animation parameters for the speech animation. The speech track also specifies the audio file for the pre-recorded speech by the <AudioFile> tag. It should be noted that the inclusion of the speech track enables the use of pre-defined or pre-recorded speech animations. Unlike the expression track, the speech track cannot be overlapping. The syntax of AFML is shown in Figure 4. Following is an example of an expression track:

```
<ExpressionsTrack name="Emotions">
 <Expression>
   <StartTime>00:00:800</StartTime>
   <ExName>smile.ex</ExName>
   <Envelope>
     <Point>
       <Shape>log</Shape>
       <Duration>0.5<Duration>
       <Int>1<Int>
     </Point>
     <Point>
       <Shape>log</Shape>
       <Duration>0.5<Duration>
```

```
        <Int>0.7<Int>
      </Point>
      <Point>
        <Shape>exp</Shape>
        <Duration>0.8<Duration>
        <Int>0<Int>
      </Point>
      </Envelope>
  </Expression>
 </ExpressionsTrack>
```

Following is an example of a speech track:

```
<SpeechTrack>
  <StartTime>00:07:600</StartTime>
  <FileName>speech1.fap</FileName>
  <AudioFile>speech1.wav</Audiofile>
</SpeechTrack>
```

3.3. Avatar Body Markup Language

ABML is very similar in format to AFML. The `<Settings>` tag is the same except that it contains one `<BAPLibPath>`, to locate the pre-defined BAP animations, in addition to the `<Fps>` and `<Duration>` tags. There can be many `<BodyAnimtionTarck>`, each specifying certain actions. The `<Mask>` tag can be used to indicate which BAPs are active for a particular track, so that the calculation of all the BAPs is not necessary. Though this tag is specific to MPEG-4 BAPs (and hence optional), it is retained for the computational advantage it offers to the underlying animation module. The body animation uses BAP files (using `<PredefinedAnimation>` tag) and behaviors (using various "Action" tags) as the basic elements for animation. One or more body animation tracks can be defined as shown in Figure 5 above. Each track has a start time, speed and priority of the action. The speed can be slow, normal, or fast. The `<Priority>` tag specifies which BAPs are to be used when several sequences overlap. An intermediate solution is automatically computed when the given priorities are equal. The `<Intensity>` tag for the pre-defined animation can be used to exaggerate or scale down the effect of the pre-defined animation.

3.4. Behaviors

The purpose of defining the behaviors is to generate, in real-time, a BAP output for gestures that cannot be predefined (e.g. a shop assistant wants to point at a particular item and tell its cost). The behaviors are embedded in the ABML as shown above. The *Facing*, *Pointing, Walking, Waiting* and *Resetting* are the basic behaviors defined. Each behavior has target location specified by the *x*, y, and z coordinates (by `<XCoor>`, `<YCoor>`, and `<ZCoor>` tags) in 3D space. Additionally, the walking behavior can specify any number of control points through which the avatar must pass, by using one or more `<ControlPoint>`

```
<BodyAnimationTrack name="Track name">
  <Mask>int[296], the BAP indices that are
  affected by this track are set to 1, the
  rest 0 - optional tag</Mask>
  <PredefinedAnimation>
    <StartTime>
      {mm:ss:mmm/autosynch/autoafter}
    </StartTime>
    <FileName>"filename.bap"</FileName>
    <Speed>{normal/slow/fast}</Speed>
    <Intensity>0 to n</Intensity>
    <Priority>0 to n</Priority>
  </PredefinedAnimation>
  <FacingAction>
    <StartTime>
      {mm:ss:mmm/autosynch/autoafter}
    </StartTime>
    <XCoor>target's X coordinate in meters
    </XCoor>
    <YCoor>target's Y coordinate in meters
    </YCoor>
    <ZCoor>target's Z coordinate in meters
    </ZCoor>
    <Speed>{normal/slow/fast}</Speed>
    <Priority>0 to n</Priority>
  </FacingAction>
  <PointingAction>
    <StartTime>
      {mm:ss:mmm/autosynch/autoafter}
    </StartTime>
    <XCoor>target's X coordinate in meters
    </XCoor>
    <YCoor> target's Y coordinate in meters
    </YCoor>
    <ZCoor>target's Z coordinate in meters
    </ZCoor>
    <Speed>{normal/slow/fast}</Speed>
    <Priority>0 to n</Priority>
  </PointingAction>
  <WalkingAction>
    <StartTime>
      {mm:ss:mmm/autosynch/autoafter}
    </StartTime>
    <ControlPoint>
    <XCoor> target's X coordinate in meters
    </XCoor>
    <ZCoor> target's Z coordinate in meters
    </ZCoor>
    </ControlPoint>
    <Speed>{normal/slow/fast}</Speed>
    <Priority>0 to n</Priority>
  </WalkingAction>
  <ResettingAction>
    <StartTime>
    {mm:ss:mmm/autosynch/autoafter}
    </StartTime>
    <Speed>{normal/slow/fast}</Speed>
    <Priority>0 to n</Priority>
  </ResettingAction>
</BodyAnimationTrack>
```

Figure 5. ABML Syntax

tags. Figure 6 below shows an example of ABML containing pre-defined animation as well as behaviors.

Since the duration of parametric actions may not be known at design time, a "synchro mode" has been added to the ABML as an alternative to the starting time parameter, which allows actions to be triggered relative to each other: “autosynch”, in place of the start time, indicates that the action will be triggered at the same time as the preceding action, “autoafter”, in place of the start time, indicates that the action will be triggered directly after the preceding action has completed. There are no limitations to the possible combinations of start time/autosynch/autoafter.

```
<ABML>
  <Settings>
    <Fps>25</Fps>
    <Duration>00:10:000</Duration>
    <BALibPath>.\BAPFiles\</BALibPath>
  </Settings>
  <BodyAnimationTrack name="Dance">
    <PredefinedAnimation>
      <StartTime>00:00:000</StartTime>
      <FileName>charleston.bap</FileName>
      <Speed>normal</Speed>
      <Intensity>1</Intensity>
      <Priority>1</Priority>
    </PredefinedAnimation>
  </BodyAnimationTrack>
  <BodyAnimationTrack name="FacedPoint">
    <FacingAction>
      <StartTime>00:02:000</StartTime>
      <XCoor>-10</XCoor>
      <YCoor>-10</YCoor>
      <ZCoor>-10</ZCoor>
      <Speed>normal</Speed>
      <Priority>2</Priority>
    </FacingAction>
    <PointingAction>
      <StartTime>autoafter</StartTime>
      <XCoor>45</XCoor>
      <YCoor>10</YCoor>
      <ZCoor>10</ZCoor>
      <Speed>normal</Speed>
      <Priority>2</Priority>
    </PointingAction>
  </BodyAnimationTrack>
</ABML>
```

Figure 6. ABML Example

4. AML Processing

Figure 7 gives the details of the specific AML processor that will be integrated in the MPEG-4 compliant real-time animation platform. All the associated software modules have been implemented and are under testing and further development for real time animation, to be plugged into the SoNG player.

The following subsections explain the various elements required for the AML processing.

4.1. AML Processor

The AML processor synchronizes all the peripheral modules, shown in Figure 7, and described below. It accepts an AML script, parses it and makes appropriate calls to the TTS Engine, AFML parser and the ABML parser with corresponding data extracted from the AML as input. The AML processor obtains the phoneme information from TTS. It then inserts it into the AFML with appropriate timing information, before calling the AFML parser. This facilitates mixing of facial expressions and the lip movement information that is obtained from TTS. Finally, the resulting audio and FAPs/BAPs from these modules are passed on to the Audio and FBA encoders respectively. The resulting encoded streams are encapsulated into an MPEG-4 complaint bit stream, ready to be transmitted over the network.

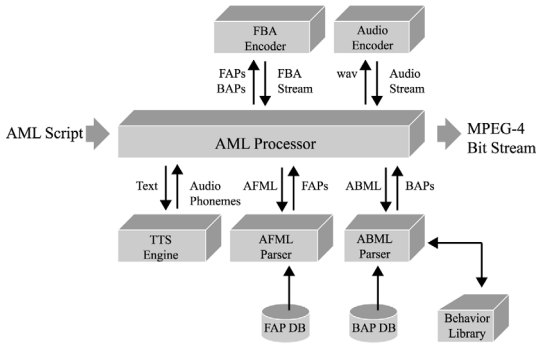

Figure 7. AML Processing Scheme

4.2. TTS Engine

The TTS engine, in general, receives text as input from the AML Processor and outputs a combination of an audio stream and phoneme sequence with timing information. Some specific implementations (as the one used in SoNG project) are able to output an MPEG-4 fap file or a viseme file (representing lip movements), in addition to the phoneme sequence. Both these files contain the timing information about speech and facial expressions for an animation sequence. Further, it may be possible to annotate the input text with emotional tags. These tags should affect the audio as well as the resulting faps, providing expressive speech as well as facial animation. AML does not specify the format for the expression tags embedded in text, but any format acceptable by the underlying TTS engine in use is

valid. The text within the <Text> tag of the <TTS>, is directed to the TTS engine, and thus the AML does not put any restrictions on its format. Microsoft SAPI tags [13] or MPEG-4 defined TTS bookmarks [5] are two such possibilities.

4.3. AFML Parser

The AFML parser receives AFML as input and generates FAP streams by merging a number of expression tracks. Expression tracks are based on predefined animation units stored in the FAP Database (FAP DB). However, the parameters such as duration, intensity and time envelopes *etc.* can be varied using the tags of the AFML. AFML can also include a sequence of phonemes (a viseme file generated by TTS), which can be converted into lip animation by the AFML parser. Co-articulation for speech animation, blending between various expressions activated at the same time, and mixing between speech and expressions are the important tasks of the AFML parser to generate smooth animations. The parser reads the AFML from either a file or a text stream. It generates either a FAP file to be used for animation later, or returns FAPs frame by frame when requested.

4.4. FAP DB

The AFML uses various animation units stored in the FAP DB. As explained in Subsection 2.2, there are 66 low-level FAPs and 2 high level FAPs. All the definition files for the high level FAPs (for 14 visemes and 6 expressions) form part of the FAP DB. In addition, there are several other "non-standard" expression files useful to add variety to animations, *e.g.* confused, sleepy *etc*. Thus, it is very easy to extend the database and refer the newly defined expressions in the AFML by the filenames without any need to have any additional tags. The same is true with the ABML and BAP DB.

4.5. ABML Parser

Similar to AFML, ABML is parsed by an ABML parser, which generates BAP streams. The ABML parser is capable of working with a number of body animation tracks containing predefined BAP files stored in the BAP DB or it can also generate body animations from parameterized behavior calls. The parser provides a way to organize bodily behaviors (gestures and postures) using timing information relative to each other (e.g. simultaneous or temporally serial), and allows the real-time modification of the animation's speed, intensity and extent (using masks which can inhibit some part of the skeleton). It is also computes intermediate skeleton states for several gestures (morphing) when overlapping specifications are given, which users can control by assigning different priorities to each action.

4.6. BAP DB

The BAP DB contains a number of body animation files that are the basic building blocks used by the ABML Parser. These MPEG-4 animations can be generated using motion capture techniques. They can also be manually designed using dedicated authoring software such as 3DS Max and appropriate exporter plug-ins developed within the SoNG project. As explained previously, the body animation in MPEG-4 is defined in terms of 296 Body Animation Parameters. A bap file, like a fap file, contains fare-by-frame bap information for an animation, in addition to the information such as frame rate and length of animation.

4.7. Behavior Library

The *Behavior Library* is triggered by the ABML parser. The C++ based library makes use of classic animation techniques based on matrix manipulation (e.g. quaternion interpolation) and of IKAN, an inverse kinematics library developed at the University of Pennsylvania [12]. The modular structure of the architecture and extensive use of powerful C++ features such as polymorphism, make it very easy to add new behaviors and greatly facilitate the reuse of already developed behaviors. *E.g.* the walking behavior automatically triggers several facing and predefined animations, thus it is very simple and easily maintainable. The ABML parser processes (*e.g.* mixes with predefined animations, changes speed *etc.*) the animations generated by the Behavior Library transparently, just like any other predefined animation.

4.8. FBA and Audio Encoders

FAPs and BAPs extracted by the AML Processor from the AFML and ABML parsers are passed to the Face and Body Animation (FBA) encoder. The output of this module is binary FBA data that merges the animations generated independently by the aforementioned modules. The FBA output is subdivided into Access Units (AU) each associated with a BIFS-anim compliant header. It is compressed using the Discrete Cosine Transform (DCT). In accordance with the MPEG approach, some AU can be encoded in an Intra mode (all values quantized) while others can be stored as predictive frames (the relative frame to frame change is quantized). Similarly the encoded audio stream along with the FBA stream can be used to transmit the animations over an MPEG-4 compliant network.

In the current implementation, the FAPs and BAPs are finally encoded and transmitted across the network from server to client. However, it is possible that AML itself is used for communication and the AML processor is implemented on each client. Though AML was not designed for this particular purpose, this will reduce the bandwidth requirements. However, this puts two requirements on the client, availability of the FAP

DB and the BAP DB and implementation of the AML processor.

5. Discussion and Conclusion

After explaining the AML in its full details, it is time to compare AML with VHML [9], mentioned in Section 1. VHML proposes to encapsulate various other markup languages to describe face and body animations, TTS and dialogue management attributes. Still under development, it does not provide support to body animations yet, and especially behavioral animations. For facial animation and speech, the biggest difference between AML and VHML is the lack of time sychronization information in VHML. This information is vital for seamless animation. Consider the following example in VHML:

```
<happy duration="7s''/>
     It's my birthday today.
```

This would lead to the virtual human showing happy emotion for a duration of 7 seconds while the sentence is being said. However, if one wants to only partially overlap speech and expression, this format is not sufficient. For example, the avatar starts speaking, and the happy expressions is dealyed slightly. Further, the happy expression can continue even after the speech is over. This effect will be achived by AFML by the following:

```
<ExpressionsTrack name="Emotion">
  <Expression>
    <StartTime>00:00:800</StartTime>
    <ExName>"happy"</ExName>
    <Envelope>
      <Point>
        <Shape>linear</Shape>
        <Duration>3.5<Duration>
        <Int>1<Int>
      </Point>
      <Point>
        <Shape>linear</Shape>
        <Duration>3.5<Duration>
        <Int>0<Int>
      </Point>
    </Envelope>
  </Expression>
</ExpressionsTrack>
<SpeechTrack name="birthday">
      <StartTime>00:00:000</StartTime>
      <FileName>"birthday.vis" </FileName>
      <AudioFile>"birthday.wav"</AudioFile>
</SpeechTrack>
```

Thus, mixing of various expressions can be carried out at any temporal position and with any time envelope allowing lot of control over final animation. It is alternatively possible to add a TTS track in AML with the required sentence, if a TTS module is available. It is easy to notice that the addition of the detailed timing information makes AML a bit complicated as compared to VHML. However, in return we get a lot of power and flexibility to the definition of animation. VHML leaves the details of the animation to the particular implementation, such as how expressions are mixed with speech and other expresions. On the other hand, AML has possibility to encode these details making the implementation of the real-time animation module a lot simpler. The separation between speech and various expression tracks in AML allows a flexible control over rich facial animations, rather than attaching expression tags directly to sentences. This feature also allows AML to be used as a design tool for creative animators. Further, any high level user defined expression can be used in AFML (including the emotions and facial movements like look-left, eyes-down) as opposed to only pre-defined tags in VHML. All the commonly used expressions are defined in the FAP DB, and more can be defined and referred in AFML thus allowing a lot of variety in animations. The same is the case with the BAP DB and the ABML. Though we have used MPEG-4 FAPs and BAPs as parameters, the language itself is not restricted by any such specification.

In conclusion, we have proposed a new animation language describing complete avatar animation consisting of face and body animation and text to speech as the basic elements. The important features of AML are:

- It is a high-level language easy to be used by the intelligent agents driving 3D avatar geometries.
- It is also easy to use by animators to create animations dynamically. It is flexible enough to create a rich variety of animations, even though it uses basic pre-defined animation blocks. It is not restricted by a fixed set of expression/gesture tags, but can be easily extended.
- It allows seamless animations using expressions, gestures and TTS, and also gives explicit control over their mutual synchronization.
- It is independent of the underlying low-level animation parameters, and hence the designers of various avatar based systems are free to choose their own implementations, but still able to share the animations.

Though we have addressed only avatar animation in this paper, it would further be interesting to recognize the user's speech, facial expressions and body postures automatically and convert directly into AML. An intelligent interactive agent can then use this

information not only to drive the avatar, but also to formulate appropriate responses and reactions.

6. Acknowledgements

The European IST Project 10192 SoNG has supported the work described in this paper. We are grateful to all the partners of the SoNG project for their direct and indirect involvement and help. Many thanks to Yasmin Arafa for providing inspiration for this work. Finally, special thanks are due to Chris Joslin for proof reading the manuscript.

References

1. K. Perlin, and A. Goldberg, “Improv: A System for Scripting Interactive Characters in Virtual Worlds”, *Proceedings of SIGGRAPH 96*, ACM Press, pp. 205-216.
2. J. Cassell, H. Vilhjalmsson, and T. Bickmore, “BEAT: the Behavior Expression Animation Toolkit”, *Proceedings of SIGGRAPH 01*, ACM Press, pp. 477-486.
3. Microsoft Virtual Worlds, http://www.vworlds.org/
4. Blaxxun Interactive, http://www.blaxxun.com/
5. ISO/IEC JTC 1/SC 29/WG11 N2502, Information Technology – Generic Coding of Audio-Visual Objects, Part 2: Visual, October 1998.
6. ISO/IEC JTC 1/SC 29/WG11 N2739 subpart 2, MPEG-4 Version 2- BIFS, March 1999.
7. S. Kshirsagar, T. Molet, N. Magnenat-Thalmann, “Principal Components of Expressive Speech Animation”, *Proceedings Computer Graphics International 2001*, IEEE Computer Society, July 2001, pp 38-44.
8. R. Boulic, P. Becheiraz, L. Emering, and D. Thalmann, “Integration of motion control techniques for virtual human and avatar real-time animation”, *Proceedings ACM Symposium on Virtual Reality Software and Technology 1997*, ACM Press, September 1997, pp. 111-118.
9. http://www.vhml.org/
10. S. Kshirsagar, M. Escher, G. Sannier, N. Magnenat-Thalmann, "Multimodal Animation System Based on the MPEG-4 Standard", *Proceedings Multimedia Modelling 99*, Ottawa, Canada, October 1999, World Scientific Publishing, pp. 215-232.
11. http://www.h-anim.org/
12. D. Tolani, A. Goswami, and N. Badler: "Real-time inverse kinematics techniques for anthropomorphic limbs." *Graphical Models* 62 (5), Sept. 2000, pp. 353-388.
13. Microsoft Speech Technologies, http://www.microsft.com/speech