Player Markup Language

PML 2.1
Specification

Yvonne A. Jung
Fraunhofer IGD

December 16, 2009

Contents

1 Player Markup Language (PML) 4
1.1 PML documents 4
1.2 PML handling 4
1.3 PML identifiers 5
1.4 PML examples 5
1.4.1 1st Example: definitions 5
1.4.2 2nd Example: actions 6

2 Top level element <definitions> 8
2.1 Element <repository> 8
2.2 Element <undefine> 9
2.3 Element <character> 10
2.3.1 Element <voice> 11
2.3.2 Element <viseme> 11
2.3.3 Element: <soundSource> 12
2.3.4 Element: <canvas> 12
2.3.5 Element: <fragment> 12
2.3.6 Element: <target> 13
2.3.7 Element: <complexion> 13
2.3.8 Element: <singlePose> 14
2.3.9 Element: <implicitPose> 14
2.3.10 Element: <multiPoses> 14
2.3.11 Element: <createSinglePose> 15
2.3.12 Element: <createMultiPoses> 16
2.3.13 Element: <locomotion> 16
2.3.14 Element: <idlePoses> 17
2.4 Element: <object> 18
2.4.1 Element: <guiContainer> 19
2.4.2 Elements: <audio>, <image>, <video> 19
2.4.3 Element: <menu> 19
2.4.4 Element: <slider> 20
2.4.5 Element: <camera> 20

3 Top level element: <actions> 22
3.1 Element: <character> 22
3.1.1 Element: <show> 23
3.1.2 Element: <hide> 23
3.1.3 Element: <transform> 24
3.1.4 Element: <startIdleList>, <stopIdleList> 24
3.1.5 Element: <animate> 25
3.1.6 Element: <stopAnimate> 26
3.1.7 Element: <complexion> 27
3.1.8 Element: <speak> 27
3.1.9 Element: <text> 28
3.1.10 Element: <audio> 29
3.1.11 Element: <pause> 29
3.2 Element: <object> 30
3.2.1 Element: <animate> 31

3.2.2 Elements: <assignImage>, <assignText>, <startAudio>, <stopAudio>, <startVideo>, <stopVideo> . . . 32
3.2.3 Element: <assignMenu> . . . 32
3.2.4 Element: <selectMenu> . . . 33
3.2.5 Element: <assignSlider> . . . 33
3.2.6 Element: <startFollowing> . . . 34
3.2.7 Element: <stopFollowing> . . . 34
3.2.8 Element: <activateEffect> . . . 34
3.2.9 Element: <deactivateEffect> . . . 34
3.2.10 Element: <zoom> . . . 35
3.2.11 Element: <frameTarget> . . . 35
3.3 Element: <schedule> . . . 36

4 Top level element: <message> 37

5 Top level element: <query> 39

6 Grammar 40
6.1 PML2 document . . . 40
6.1.1 Definitions . . . 40
6.1.2 Actions . . . 42
6.1.3 Messages . . . 44
6.1.4 Queries . . . 45
6.2 Basic grammar . . . 45

7 Annex – Controlling X3D with PML 47
7.1 Notes . . . 47
7.2 Introducing an Animation Control Language . . . 47
7.3 Handling Animations with PML . . . 48
7.4 Scheduling and Controlling Animations . . . 50
7.4.1 Timeline and PML-Processor . . . 50
7.4.2 AnimationController and AnimationContainer . . . 50

References 52

1 Player Markup Language (PML)

PML is based on XML and is a result from the Virtual Human project [1], which is used and extended within the ANSWER project [2]. It is a control language for virtual characters and other objects. Thus, it's possible to control interactive scenes easily.

The PML-Script describes a scene act. The PML-Script relates to one virtual character, which is referenced inside the script by the *id-Field*. The script schedules the movements of the virtual character. These movements can run successively or simultaneously.

PML distinguishes between *definition-scripts* and *actions-scripts*. The *definition-scripts* describe the animations, which are available for a character. Animations in the other *actions-scripts* should be referenced in the *definition-script*.

At the beginning of an application, the definition-scripts have to be loaded. Within the animation-scripts, it is then possible to refer the animations of the characters. The actions-scripts defines the temporal sequence of the animations.

1.1 PML documents

Each PML document has exactly one of the following top-level elements, which are defined and described in more detail in the following sections. Furthermore, each document must contain a valid XML-Header:
<?xml version="1.0" encoding="ISO-8859-1"?>

- <definitions> for the object and character definitions
- <actions> for object and character definitions
- <message> for message exchange with the players
- <query> for queries to the player

Grammar:

```
PML2DOC ::= DEFINITIONS|ACTIONS|MESSAGE|QUERY
```

1.2 PML handling

The processing of PML-scripts is controlled via messages. For the corresponding commands and status messages the following rules apply:

- If the player receives either a definition-script or a actions-script, he send the status **fetched**. In a message script or query script, no fetched will be transmitted.
- Actions-scripts must be started explicitly. All other scripts starts automatically after they were received. Actions-scripts can be started in two ways:
  - start command in a message
  - start attribute in actions-scripts
  - As soon as the first action in a actions-script runs the player sends back the status **started**.
- If an actions-script will be terminated prematurely due to the stop command, the player sends the status **stopped**.
- If the player fails to execute an action, he sends the status **failed**. The reference is the id of the failed script, and (optionally) an error message.

[1]http://www.virtualhuman.de/
[2]http://www.answer-project.org/

```
<message id="avl_13_s">
    <state refId="mmac1" type="failed" description="Multipose fingerring1.x3d not defined."/>
</message>
```

- If a definitions-script or actions-script is fully processed, the player sends the **finished** status. In a message or query script no status will be returned.
- To get all modules back to their initial state, a **reset** message will be send.

```
<message id="msg000">
    <command refId="scene" type="reset"/>
</message>
```

  This message causes the player to discard all definitions and abort all running actions. After this the player sends a message with the **finished** state back. The identifier in the *refId* attribute will be the same as given in the message-script (*scene* for the example above) **Attention:** A message-script with the *reset* command is the only message on which the player returns a *finished* state.

- Each query in a query-script will be replied through a message-script.

Furthermore, two types of actions can be distinguished: First, there are transitions (that have no duration, i.e. `dur="0"`), which simply trigger certain events (e.g. `<show>`). And second, there are animations like a gesture, which have a duration greater than zero (e.g. `dur="1000"`).

1.3 PML identifiers

Each object which can be referenced contains a identifier (ID) which will be specified with the *id* attribute. Identifiers could be generated at runtime or assigned statically. According to their usage, they must be unique over the entire system or just in their current context. They can be referenced through the *refId* attribute.
The following definition is not part of the languange definition: The ID is an alpha-numeric string, which consists of a module identifier, a seperator *::* and a number. Each module is responsible for generating unique IDs in the according namespace. Examples for valid identifiers are: "'ne::1"', "'cde::123"', "'av::12345"'

1.4 PML examples

1.4.1 1st Example: definitions

Building visemes and emotions from morphtargets.

```
<definitions id="Bsp1">
  <repository id="myrep">
    <path src="http://www.vitualhuman.org/data/mychar">
  </repository>

  <character id="My Character" src="rep://myrep/character.wrl">
    <!-- Facial muscles -->
    <singlePose id="Neutral"            src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="L_Eyebrow_Up"       src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="R_Eyebrow_Up"       src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="L_Eyebrow_Down"     src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="R_Eyebrow_Down"     src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="L_Mouthcorner_Up"   src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="R_Mouthcorner_Up"   src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="L_Mouthcorner_Down" src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="R_Mouthcorner_Down" src="rep://myrep/myCharacter.wrl" dur="2000" />
```

```
    <singlePose id="Upperlip_Up"        src="rep://myrep/myCharacter.wrl" dur="2000" />
    <singlePose id="LowerLip_Down"      src="rep://myrep/myCharacter.wrl" dur="2000" />

    <singlePose id="Phon_M" src="rep://myrep/Phon_MShape.wrl" dur="2000"/>

    <!-- Ekman's basic emotions -->
    <createSinglePose id="Emot_Angry">
      <singlePose refId="L_Eyebrow_Down"     intensity="0.7" />
      <singlePose refId="R_Eyebrow_Down"     intensity="0.7" />
      <singlePose refId="L_Mouthcorner_Down" intensity="0.7" />
      <singlePose refId="R_Mouthcorner_Down" intensity="0.7" />
      <singlePose refId="Neutral"            intensity="-1.8" />
    </createSinglePose>

    <!-- A more complex emotion -->
    <createSinglePose id="Emot_Disliking">
      <singlePose refId="Emot_Disgust"       intensity="0.6" />
      <singlePose refId="Neutral"            intensity="0.4" />
    </createSinglePose>

    <!-- Visemes -->
    <createSinglePose id="Phon_A">
      <singlePose refId="Upperlip_Up"        intensity="0.2" />
      <singlePose refId="LowerLip_Down"      intensity="0.5" />
      <singlePose refId="L_Mouthcorner_Down" intensity="0.3" />
      <singlePose refId="R_Mouthcorner_Down" intensity="0.3" />
      <singlePose refId="Neutral"            intensity="-0.3" />
    </createSinglePose>
    <createSinglePose id="Phon_E">
      <singlePose refId="Upperlip_Up"        intensity="0.2" />
      <singlePose refId="LowerLip_Down"      intensity="0.5" />
      <singlePose refId="L_Mouthcorner_Up"   intensity="0.1" />
      <singlePose refId="R_Mouthcorner_Up"   intensity="0.1" />
      <singlePose refId="Neutral"            intensity="0.1" />
    </createSinglePose>
  </character>
</definitions>
```

1.4.2 2nd Example: actions

```
<actions id="ac1" start="false">
  <character refId="character1">
    <animate id="a1" dur="1000" alignTo="null" alignType="null">
      <gesture refId="move" />
      <multiPoses refId="anim1" />
    </animate>
    <speak id="s1" dur="1500" alignTo="a1" alignType="starts-with">
      <text>Hello Folks</text>
      <audio src="http://a.server.de/hello.wav">
        <phoneme refId="he" dur="100" />
        <phoneme refId="lo" dur="100" />
```

```
      <phoneme refId="_"  dur="200" />
      <phoneme refId="fo" dur="75"  />
      <phoneme refId="ks" dur="63"  />
    </audio>
  </speak>
</character>
<object refId="object1" >
  <animate id="a2" refId="move" dur="2000" alignTo="a3" alignType="after" />
    <multiPoses refId="anim2" />
  </animate>
  <show id="a3" alignTo="null" alignType="null" />
</object>
<schedule>
  <par>
    <action refId="a1" begin="0" dur="1000" />
    <action refId="s1" begin="0" dur="1500" />
    <seq>
      <action refId="a3" begin="0" dur="0" />
      <action refId="a2" begin="0" dur="2000" />
    </seq>
  </par>
</schedule>
</actions>
```

2 Top level element <definitions>

All characters, 3D-objects and animations, which will be used in specific actions, must be defined in advance. Scene definitions are used to tell the player which scene elements will be used in future scripts, and how the associated data can be loaded. Every scene element which will be used in the scripts has to be registered and labeled.

Every child element from *<definition>*, which has a graphical representation is not visible after loading. Neither animation nor media are active at this time. Definition-scripts can be sent to the player at any time. They will be executed by the player immediately after receiving. All required data will be immediately loaded into memory, and all scene elements will be instantiated and registered immediately.

Attributes:

- id - Identifier.

child elements:

- <repository> (optionally): Search path.
- <undefine> (optionally): Delete scene object.
- <character> (optionally): Character definitions.
- <object> (optionally): Object definitions.

Grammar:

```
DEFINITIONS ::= <definitions id="ID">
                 REPOSITORY*
                 UNDEFINE*
                 (CHARACTER|OBJECT)*
                </definitions>
ID          ::= [a-zA-Z0-9.:_-]+
```

2.1 Element <repository>

A repository is a collection of search paths, which can be used to find files. Three different protocols were supported: *file*, *http* and *rep*.
The protocol *rep* is used to access existing repositories. The name of the repository is given by the string between "'rep://"' and the first "'/"'. It is not allowed to access a repository within another repository.
The paths will be searched in the given order.

Attributes:

- id: identifier

Child elements:

- <path>: search path

Attributes of the child elements:

- src: file path

Grammar:

```
REPOSITORY ::= <repository id="ID">
                PATH+
              </repository>
PATH ::= <path src="SRCSIMPLE" />
```

SRC ::= (file|http|rep)://SRCSEGMENT

SRCSIMPLE ::= (file|http)://SRCSEGMENT

SRCSEGMENT ::= (SRCSEGMENT/SRCSEGMENT)|ALPHANUMPATH+

ALPHANUMPATH ::= [a-zA-Z0-9:.-_]

Example:

```
<definitions id="definition0">
...
  <repository id="test">
    <path src="file://C:/VH_SOFTWARE/VirtualHuman/bin/" />
    <path src="http://www.data.org/xsd/" />
  </repository>
...
</definitions>
```

2.2 Element <undefine>

With the <undefine> element already loaded scene objects can be removed from the memory of the player. The scene object to remove is specified by the *refID attribute*. It is not allowed to reference an already deleted object.

Attributes:

- refId - Reference to the scene object to remove.

Grammar:

UNDEFINE ::= <undefine refId="ID" />

Example:

```
<definitions id="undef">
  <undefine refId="Susan" />
</definitions>
```

2.3 Element <character>

This element is a virtual character with his gestures, mimic skills, appearance and voice. The character is labeled with a global unique name. With this name the virtual character can be referenced within other PML commands. Each of the characters animation is labeled with a *id* which is unique within the character scope.

Attributes:

- id - Unique identifier.
- src - Reference to the file that contains the basic geometry of the character.
- root (optionally) - Name of the node in the file which defines the character. If it is not defined, the first node of the suitable type in the file will be used.

child elements:

- <voice>(optionally): Voice of the character
- <viseme> (optionally): Mapping from Phonemen to Viseme.
- <soundSource> (optionally): Position of the voice.
- <canvas> (optionally): Areas whose texture can be adjusted.
- <fragment (optionally): Definition of modifiable parts.
- <target> (optionally): Expression of implicitPose targets.
- <complexion> (optionally): Definitions for face textures.
- <singlePose> (optionally): Description of a character state.
- <implicitPose> (optionally): Animations which are generated at runtime.
- <multiPoses> (optionally): Predefined animations.
- <createSinglePose> (optionally): Used to create a new singlePose through already existing.
- <idlePoses> (optionally): Background animations to be played when the character has nothing to do (idle).
- <createMultiPoses> (optionally): Used to create new multiPoses through already existing.[3]
- <locomotion> (optionally): Used to create compound multiPoses for locomotion through already existing.[4]

Grammar:

```
CHARACTER ::= <character id="ID" src="SRC" [root="ID"]>
                VOICE 0,1
                VISEME 0,1
                SOUNDSOURCE 0,1
                (CANVAS|FRAGMENT|TARGET|COMPLEXION|SINGLEPOSE|MULTIPOSES|
                 IMPLICITPOSE|CREATESINGLEPOSE|IDLEPOSES|CREATEMULTIPOSES|LOCOMOTION)*
              </character>
```

[3]PML 2.1
[4]PML 2.1

2.3.1 Element <voice>

Attributes:

- id: Name of the voice.
- refId: Location of the voice (Reference to soundSource).
- pitch: Standard pitch of the voice.
- range: Standard pitch of the voice area.
- rate: Standard voice rate.
- volume: Standard volume.

Grammar:

```
VOICE       ::= <voice id="ID" refId="ID" pitch="PITCHLEVEL"
                 range="PITCHLEVEL" rate="RATELEVEL" volume="VOLUMELEVEL" />
PITCHLEVEL  ::= default|x-low|low|medium|high|x-high
RATELEVEL   ::= default|x-slow|slow|medium|fast|x-fast
VOLUMELEVEL ::= silent|x-soft|soft|medium|loud|x-loud
```

Example:

```
<definitions id="definition0">
  ...
  <character id="valerie" src="file://C:/VirtualHuman/Characters/valerie.wrl">
    ...
    <voice id="stimme" refId="marlene16D" pitch="default"
    range="high" rate="fast" volume="soft"/>
    ...
  </character>
</definitions>
```

2.3.2 Element <viseme>

Describe the mouth movement of a character.

Child elements:
<phoneme>: Mapping from phonemes to viseme. Phonemes describe the hearable characteristics of a language.

Attributes of the child elements:

- id: Name of phonemes.
- refId: Referring to a viseme (singlePose).
- intensity: Intensity of the referenced singlePose.

Grammar:

```
VISEME ::= <viseme>
             PHONEME+
           </viseme>
PHONEME ::= <phoneme id="ID" refId="ID" intensity="FLOAT" />
```

Example:

```
<definitions id="definition0">
  ...
  <character id="valerie" src="rep://valerie/valerie.wrl">
    ...
    <viseme>
      <phoneme id="eh" refId="myvisem_eh" intensity="0.5" />
      <phoneme id="en" refId="myvisem_en" intensity="1.0" />
    </viseme>
    ...
  </character>
</definitions>
```

2.3.3 Element: <soundSource>

The <soundSource> element defines locations in the scene, which can be used as a source for sounds.

Attributes:

- id: Unique identifier.
- src: (optionally) Reference to the appropriate file. If *src* is not specified, *src* from object will be used.
- root: (optionally) Name of the node in the file. Without specifying, the first node of the suitable type will be used.

Grammar:

```
SOUNDSOURCE ::= <soundSource id="ID" [src="SRC"] [root="ID"]/>
```

2.3.4 Element: <canvas>

Specifies where textures can be applied.

Attributes:

- id: Unique identifier.
- src: (optionally) Reference to the appropriate file. Without specifying *src* from object will be used.
- root: (optionally) Name of the node in the file. Without specifying, the first node of the suitable type will be used.

Grammar:

```
CANVAS ::= <canvas id="ID" [src="SRC"] [root="ID"] />
```

2.3.5 Element: <fragment>

The <fragment> element specifies sub-objects. It is possible to show, hide or transform/move these sub-objects.

Attributes:

- id: Unique identifier.
- src: (optionally) Reference to the appropriate file. Without specifying *src* from object will be used.
- root: (optionally) Name of the node in the file. Without specifying, the first node of the suitable type will be used.

Grammar:

```
FRAGMENT ::= <fragment id="ID" [src="SRC"] [root="ID"] />
```

Example:

```
<definitions id="definition0">
  ...
  <character id="valerie" src="./valerie.wrl">
    <fragment id="Trafo" src="./valerie.wrl" root="MyTransformNode"/>
    ...
  </character>
  ...
</definitions>
```

2.3.6 Element: <target>

The <target> element identifies possible targets of implicitPose.

Attributes:

- id: Unique identifier.
- root: Name of the node in the character file specifications.

Grammar:

```
TARGET ::= <target id="ID" root="ID" />
```

2.3.7 Element: <complexion>

With the <complexion> element the available skin textures of a character can be defined.

Attributes:

- id: Identifier of the complexion.
- refId: Reference to canvas. Indicates where the texture will be applied.
- src: Reference to the file that contains the texture.

Grammar:

```
COMPLEXION ::= <complexion id="ID" refId="ID" src="SRC" />
```

Example:

```
<definitions id="definition">
  ...
  <character id="valerie" src="rep://valerie/valerie.wrl">
    ...
    <canvas id="face" src="file://valerie.wrl" root="MyShaderNode"/>
    <complexion id="green-nose" refId="face" src="file://Textures/face_green_nose.jpg"/>
    ...
  </character>
</definitions>
```

2.3.8 Element: <singlePose>

A <singlePose> element describes the state of a 3D model.

Attributes:

- id: Unique identifier.
- src: Reference to the file that contains the animation.
- root (optionally): Name of the node in the file of this animation. If this is not defined, it is the first instance of animation in the file.
- dur (optionally): Duration in ms.

Grammar:

```
SINGLEPOSE ::= <singlePose id="ID" src="SRC" [root="ID"] [dur="NATURAL"] />
```

Example:

```
<definitions id="definition0">
  ...
  <character id="sven">
    <singlePose id="angry" src="rep://sven/angry.wrl" dur="2000" />
    ...
  </character>
</definitions>
```

2.3.9 Element: <implicitPose>

Animations which are generated at runtime can be described with this element.

Attributes:

- id: Unique identifier.
- src: Reference to the appropriate file.
- type: Type of inverse kinematics.
- dur (optionally): Duration in ms.

Grammar:

```
IMPLICITPOSE ::= <implicitPose id="ID" src="SRC" type="IKTYPE" [dur="NATURAL"] />
IKTYPE       ::= lookAt|lookAtHold|lookAtRetract|eyeGazeAt|eyeGazeAtHold|eyeGazeAtRetract|
                 pointLeftHand|pointLeftHandHold|pointLeftHandRetract|
                 pointRightHand|pointRightHandHold|pointRightHandRetract
```

2.3.10 Element: <multiPoses>

Animations are defined by the <multiPoses> element. It is also possible to extract only a part of an animation using the attributes *from* and *to*.

Attributes:

- id: Unique identifier.
- src: Reference to the file that contains the animation.

- root (optionally): Name of the node in the file which defines this animation. If this is not defined, it is the first instance of animation in the file.
- from (optionally): Start value for the definition of an excerpt from an animation. The range of the value is between [0, 1]. If no starting value is specified, this is always 0.
- to (optionally): Final value for the definition of an excerpt from an animation. The range of the value is between [0, 1]. If no final value is specified, this is always 1.
- dur (optionally): Duration in ms.

Grammar:

```
MULTIPOSES    ::= <multiPoses id="ID" src="SRC" [root="ID"] [from="FLOATINTERVAL"]
                    [to="FLOATINTERVAL"] [dur="NATURAL"] />
FLOATINTERVAL ::= (0.[0-9]+)|1.0
```

2.3.11 Element: <createSinglePose>

Using <createSinglePose> helps to create a new <singlePose> element from an existing.

Attributes:

- id: Unique identifier.

child elements:

- <singlePose>: Reference to a singlePose.

Attributes of the child elements:

- refId: Name of the referenced singlePose.
- intensity: Intensity of the referenced singlePose.

Grammar:

```
CREATESINGLEPOSE ::= <createSinglePose id="ID">
                       SINGLEPOSEREF+
                     </createSinglePose>
SINGLEPOSEREF    ::= <singlePose refId="ID" intensity="FLOAT" />
FLOAT            ::= 0|[-]((0.[0-9]+)|([1-9][0-9]*[.[0-9]+]))
```

Example:

```
<definitions id="definition0">
  ...
  <character id="valerie" src="rep://valerie/valerie.wrl">
    <createSinglePose id="Emot_Dislinking">
      <singlePose refId="Emot_Disgust" intensity="0.6" />
      <singlePose refId="Neutral" intensity="0.4" />
    </createSinglePose>
    ...
  </character>
  ...
</definitions>
```

2.3.12 Element: <createMultiPoses>

Using <createMultiPoses> helps to create a new <multiPoses> element from an existing.

Attributes:

- id: Unique identifier.
- dur (optionally): Rescaled duration in ms.

child elements:

- <multiPoses>: Reference to a multiPoses element.

Attributes of the child elements:

- refId: Name of the referenced multiPoses element.
- weight: Weighting of the referenced multiPoses.

Grammar:

```
CREATEMULTIPOSES ::= <createMultiPoses id="ID" [dur="NATURAL"]>
                       MULTIPOSESREF+
                     </createMultiPoses>
MULTIPOSESREF    ::= <multiPoses refId="ID" weight="FLOAT" />
```

2.3.13 Element: <locomotion>

Using <locomotion> allows to create compound multiPoses for locomotion through already existing, which can be used within a <moveTo> action.

Attributes:

- id: Unique identifier.
- dur (optionally): Rescaled duration in ms.

child elements:

- <speed>: Contains one or more multiPoses elements. Starting from the first up to the last child element, the average speed gets increased from $minSpeed\frac{m}{s}$ to $maxSpeed\frac{m}{s}$, which can be changed via the “minSpeed” and “maxSpeed” attributes.
- <turnAngle>: Contains one or more multiPoses elements. Starting from the first up to the last child element, the average turning angle gets increased from $maxLeft$ to $maxRight$ degrees around the up-axis (usually y, with z facing forward), which can be changed via the “maxLeft” and “maxRight” attributes (both in $[0; \frac{\Pi}{2}]$).

Children of the child elements:

- <multiPoses>: Reference to a multiPoses element, with the attributes as defined in <createMultiPoses>.

Grammar:

```
LOCOMOTION ::= <locomotion id="ID" [dur="NATURAL"]>
                 <speed minSpeed="FLOAT" maxSpeed="FLOAT">
                   MULTIPOSESREF+
                 </speed>
                 <turnAngle maxLeft="FLOAT" maxRight="FLOAT">
                   MULTIPOSESREF+
                 </turnAngle>
               </locomotion>
```

2.3.14 Element: <idlePoses>

The <idlePoses> element defines a list of animations, which will be played when a character is idle. With the random attribute the playing order of the animations can be set. So the list can be played randomly or in the predefined order.

Attributes:

- id: Unique identifier.
- random: Play order (false = specified order; true = random sequence).

child elements:

- <singlePose>: Reference to singlePose.
- <multiPoses>: Reference to multiPoses.
- <implicitPose>: Reference to implicitPose.

Attributes of child elements:
Element: singlePose

- refId: Name of the referenced singlePose.
- intensity: Intensity of the referenced singlePose.
- dur: Duration in ms.

Elements: implicitPose, multiPoses

- refId: Name of the referenced animation.
- target (only implicitPose): The target of the referenced implicitPose.
- dur (optionally): Duration in ms.

Grammar:

```
IDLEPOSES ::= <idlePoses id="ID" random="BOOL">
                POSEDURREF+
              </idlePoses>


POSEDURREF ::= <singlePose refId="ID" intensity="FLOAT" dur="NATURAL" />
             | <implicitPose refId="ID" target="ID" [dur="NATURAL"] />
             | <multiPoses refId="ID" [dur="NATURAL"] />

NATURAL ::= 0|([1-9][0-9]*)
```

Example:

```
<definitions id="definition0">
  <character id="valerie" src="rep://valerie/valerie.wrl">
    <idlePoses id="b1" random="true">
      <multiplePoses refId="jump">
      <multiplePoses refId="make-yoga">
      <multiplePoses refId="go-home">
    </idlePoses>
    ...
  </character>
  ...
</definitions>
```

2.4 Element: <object>

The <object> element defines the scene-objects which have to be loaded. Further, objects can be switched between visible and invisible, and can be animated. Furthermore, an object can play videos, show pictures, play audio files or can be treated as an element of the user interface. The 3D objects must support these features (depends on the implementation). It is the responsibility of the PML-author, that only supported objects will be assigned.
Note: Elements already defined in the previous section are not further explained here again.

Attributes:

- id: Unique identifier.
- src: Reference to the file that contains the basic geometry of the object.
- root (optionally): Name of the node in the file, which defines the object. If it is not defined, it is the first node of the suitable type (see Implementation) in the file.

child elements:

- <canvas> (optionally): Areas whose texture can be adjusted.
- <fragment> (optionally): Definition of modifiable sub-areas.
- <guiContainer> (optionally): Areas where GUI elements can be displayed.
- <soundSource> (optionally): Definition of a sound source.
- <target> (optionally): Specify ImplicitPose targets.
- <singlePose> (optionally): Description of a object state.
- <implictPose> (optionally): Animations which were generated at runtime.
- <multiPoses> (optionally): Predefined animations.
- <createSinglePose> (optionally): To create new singlePoses from existing.
- <idlePoses> (optionally): Background animations, which will be played if the object has nothing to do.
- <audio>, <image>, <video> (optionally): Multimedia data.
- <menu> (optionally): Menu to interact with the user.
- <slider> (optionally): Slider to interact with the user.
- <camera> (optionally): Defines a camera or viewpoint respectively.[5]

Grammar:

```
OBJECT ::= <object id="ID" src="SRC" [root="ID"]>
             (CANVAS|FRAGMENT|GUICONTAINER|SOUNDSOURCE|TARGET|SINGLEPOSE|IMPLICITPOSE|
              MULTIPOSES|CREATESINGLEPOSE|IDLEPOSES|MEDIA|MENU|SLIDER|CAMERA)*
           </object>
```

[5]PML 2.1

2.4.1 Element: <guiContainer>

The ability to interact with the scene is defined by the <guiContainer> element.

Attributes:

- id: Unique identifier.
- src (optionally): Reference to the appropriate file. If *src* is not specified, the one from object will be used.
- root (optionally): Name of the node on which the GUI object will be displayed.
- device (optionally): Name of the node to interact with.
- visualization: Indicates whether it is a three - or two-dimensional GUI element.

Grammar:

```
GUICONTAINER      ::= <guiContainer id="ID" [src="SRC"] [root="ID"]
                        [device="ID"] visualization="VISUALIZATIONTYPE" />
VISUALIZATIONTYPE ::= 2D|3D
```

Example:

```
<definitions id="definition0">
  ...
    <object id="Box" src="file://Object.wrl">
        ...
        <guiContainer id="user" src="file://file.wrl" root="screen"
         device="menu" visualization="3D" />
    </object>
  ...
</definitions>
```

2.4.2 Elements: <audio>, <image>, <video>

These elements can be photos, videos or audio data. These elements are only useable as child elements of object.

Attributes:

- id: Unique identifier.
- refId: Refers to canvas for image and video data and on soundSource for audio data.
- src: Reference to the appropriate media file.

Grammar:

```
MEDIA ::= <(audio|image|video) id="ID" refId="ID" src="SRC"/>
```

2.4.3 Element: <menu>

With this element a menu will be assigned to an object. The parameters are defined in an action-script.

Attributes:

- id: Unique identifier.
- refId: Refers to guiContainer.
- multi: Multiple or single selection

- src: Reference to the appropriate file (On the definition of GUI components in 2D/3D).

Grammar:

```
MENU ::= <menu id="ID" refId="ID" multi="BOOL" src="SRC" />
```

2.4.4 Element: <slider>

This element assigns a slider to the object. The parameters of the slider are defined in an actions-script.

Attributes:

- id: Unique identifier.
- refId: Refers to guiContainer.
- min: Lower bound of the interval.
- max: Upper bound of the interval.
- src: Reference to the appropriate file (on the definition of GUI components in 2D/3D).

Grammar:

```
SLIDER ::= <slider id="ID" refId="ID" min="INTEGER" max="INTEGER" src="SRC" />
```

Example:

```
<definitions id="definition0">
  ...
  <object id="DaBigCube" src="http://www.virtualhuman.org/wrl/cube.wrl">
    <sound id="Ton" root="AnAudioNode"/>
    <guiContainer id="User" src="file://file.wrl" root="screen" device="menu"
     visualization="3D" />
    ...
    <audio id="sound0" refId="Ton" src="file://audio/storm.mp3" />
    <audio id="sound1" refId="Ton" src="file://audio/wind.mp3" />
    <menu id="m0" refId="User" multi="false"
     src="http://www.virtualhuman.org/gui/menu_01.ui" />
    <slider id="s0" refId="User" min="0" max="100"
     src="http://www.virtualhuman.org/gui/slider_05.ui" />
  </object>
  ...
</definitions>
```

2.4.5 Element: <camera>

This element allows to specify a camera (i.e. viewpoint). Some parameters are defined in an actions script.

Attributes:

- id: Unique identifier.
- facingDir (optionally): the (local) viewing direction of the first target.
- fov (optionally): field of view (if not specified 0.7854 is used).
- root (optionally): Name of the node in the file.
- src (optionally): Reference to the appropriate file.

Child elements:

- <targetFull>: One or more references to targets that shall be fully framed (e.g. the whole humanoid). Note: this tag is order dependent, the first target that appears here is taken as the main target in the camera etc.
- <targetCloseUp>: One or more references of close up targets (e.g. the head of the previously mentioned humanoid). Note: this tag is likewise order dependent.

Grammar:

```
CAMERA ::= <camera id="ID" [facingDir="VECTOR"] [fov="FLOAT"] [src="SRC"] [root="ID"] >
             TARGETFULL+
             TARGETCLOSE+
           </camera>


TARGETFULL ::= <targetFull id="ID" root="ID" />
TARGETCLOSE ::= <targetCloseUp id="ID" root="ID" />
```

3 Top level element: <actions>

An actions-script consists of a number of activities and assigns them within a temporal context. In actions only such scene elements which were defined previously can be used. Every action is labeled with a unique id attribute. The order of the action definitions within <actions> has no meaning. The execution of all actions-scripts will be triggered with a start-message or starts automatically when receiving the script if its start attribute is set to true. An actions-script is executed only once. After the execution it will be deleted from the player memory.

Attributes:

- id: Identifier.
- start: Indicates whether the script should start after it was successfully parsed.

child elements:

- <character> (optionally): Character actions.
- <object> (optionally): Object actions.
- <schedule> (optionally): Temporal order of actions.

Grammar:

```
ACTIONS ::= <actions id="ID" start="BOOL">
              (CHARACTERACTION|OBJECTACTION)*
              SCHEDULE 0,1
            </actions>
BOOL    ::= true|false
ID      ::= [a-zA-Z0-9.:_-]+
```

3.1 Element: <character>

The <character> element describes all actions regarding the virtual character. Actions which can be performed by characters and other objects are: show, hide, transform and manipulation of the active idleList (startIdleList, stopIdleList). Each action will be identified by the *id* attribute. For example, a gesture or idleList will be referenced with the *refId* attribute.
Furthermore, there are actions that only can be performed by a single character. These include facial expressions and gestures. These are specified by the *animate* and *speak* attributes. Additional parameters for speech synthesis can be defined by the tags within the *text attribute.*

Attributes:

- refId: Specifies the character of an action.

child elements

- <show>: Show character.
- <hide>: Hide character.
- <transform>: character position.
- <startIdleList>, <stopIdleList>: The element <idlePoses> activate or deactivate.
- <animate>: Character animation.
- <stopAnimate>: Stops animation.[6]

[6] PML 2.1

- <complexion>: Reference to a certain skin texture or coloring.
- <speak>: Speech output.
- <pause>: Do nothing.

Grammar:

```
CHARACTERACTION ::= <character refId="ID">
                      (SHOW|HIDE|TRANSFORM|IDLELIST|ANIMATECHR|COMPLEXIONACT|SPEAK|PAUSE|STOPANIMATE)+
                    </character>
```

3.1.1 Element: <show>

The <show> element can be used to show characters or objects. In case of a camera it activates/binds the camera, which can be defined via the camera tag or for very simple setups via a fragment. If the refId points to a fragment that holds a reference to a light, then the light is switched on (or off, in case of hide).

Every action is accompanied by a ALIGNMENT tag (see 3.1.5 for some important notes) that indicates the temporal order of this action related to other actions. If no other action is relevant, this will be indicated by setting the attribute *alignTo* to 0.

Attributes:

- id: Identifier of the action.
- refId: Reference to fragment definition.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
SHOW      ::= <show id="ID" refId="ID" ALIGNMENT />
ALIGNMENT ::= [alignTo="null|ID"] [alignType="ALIGNTYPE"]
ALIGNTYPE ::= null|equals|before|after|meets|met-by|overlaps|
              overlapped-by|starts|started-by|during|contains|
              finishes|finished-by|starts-with|finishes-with
```

Example:

```
<actions id="action100" start="true">
  <character refId="sven">
    <show id="show78" refId="Trafo" alignTo="null" alignType="null" />
    ...
  </character>
  ...
</actions>
```

3.1.2 Element: <hide>

Characters and objects can be hidden using the <hide> element. In case of a camera it gets deactivated.

Attributes:

- id: Identifier of the action.
- refId: Reference to fragment definition.

- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
HIDE ::= <hide id="ID" refId="ID" ALIGNMENT />
```

3.1.3 Element: <transform>

Characters and objects can be positioned with this element.

Attributes:

- id: Identifier of the action.
- refId: Reference to fragment definition.
- pos (optionally): Position.
- orientation (optionally): Orientation.
- scale (optionally): Scaling.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
TRANSFORM  ::= <transform id="ID" refId="ID" [pos="VECTOR"]
               [orientation="QUATERNION"] [scale="VECTOR"] ALIGNMENT />
QUATERNION ::= VECTOR,FLOAT
VECTOR     ::= FLOAT,FLOAT,FLOAT
FLOAT      ::= 0|[-]((0.[0-9]+)|([1-9][0-9]*[.[0-9]+]))
```

3.1.4 Element: <startIdleList>, <stopIdleList>

Using *startIdleList* and *stopIdleList* can activate or deactivate *idlePoses*. By specifying *minDelay* and *maxDelay* a time interval can be specified. In this case, the player is waiting n milliseconds before the next idle gesture will be played, where n is a randomly selected value from the specified interval.

Attributes:

- id: Identifier of the action.
- refId: idlePoses Identifier.
- minDelay (optionally): The player waits at least *minDelay* milliseconds before the next idle gesture will be played.
- maxDelay (optionally): The player waits maximal *maxDelay* milliseconds before the next idle gesture will be played.
- concurrent (optionally): Indicates whether the idle gestures should be played in parallel to non-idle gestures. Default = false, that means, the idle gestures will be interrupted by explicit controlled gestures.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
IDLELIST ::= <startIdleList id="ID" refId="ID" [minDelay="NATURAL"]
              [maxDelay="NATURAL"] [concurrent="BOOL"] ALIGNMENT /> |
             <stopIdleList id="ID" refId="ID" ALIGNMENT />
```

Example:
The following command does the character blink every 1 to 2.5 seconds.

```
<startIdleList id="a1" refId="blink" minDelay="1000" maxDelay="2500"
 concurrent="true" alignTo="null" alignType="null" />
```

3.1.5 Element: <animate>

Characters and other objects will be animated with this element. The syntax for these two cases is a little bit different. For characters there is a distinction between facial animation (face), gestures (gesture) and posture (posture).

Note:
Only the POSEREF children as well as the 'id' attribute of <animate> are checked and evaluated on player side. This is due to the concept of iterative refinement: whereas on a very high level only a <gesture> of a certain type is requested, this has to be refined in an intermediate level – usually with the help of a so-called gesticon (a kind of dictionary which contains the player-specific mappings from high-level mimics or gesture requests to the concrete animation files, which certainly may vary for different visualization engines). Also this higher level stuff then is no longer needed, it is kept for consistency and propagated through the whole pipeline by only enriching, but not changing, the original PML scripts. Another example for this issue is the ALIGNMENT property, which also only is used on higher levels before concrete timing information, e.g. given by the TTS-system or the previously mentioned animation files, is added in the <schedule> block in a final step before sending the script to the player.

Attributes:

- id: Identifier of the action.
- refId : Name of the referenced animation.
- speed (optionally): Animation speed (as a factor).
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

child elements:

- <face>: face animation.
- <gesture>: gesture.
- <posture>: posture.
- <singlePose> (optionally): Reference to singlePose.
- <multiPoses> (optionally): Reference to multiPoses.
- <implicitPose> (optionally): Reference to implicitPose.
- <moveTo> (optionally): Reference to fragment.[7]

Attributes of the child elements:

Elements: face, gesture, posture (only of interest for higher level modules, not evaluated on player side)

[7] PML 2.1

- refId: Name of the referenced animation.
- intensity (only face): Intensity of the facial expression.

Elements: singlePose, implicitPose, multiPoses, moveTo (usually only of interest for lower level modules)

- refId: Name of the referenced animation.
- intensity (only singlePose): Intensity of the referenced singlePose.
- target (only implicitPose): The aim of the referenced implicitPose.
- pos (optionally, only moveTo): Position, analogous to transform.
- orientation (optionally, only moveTo): Orientation, analogous to transform.
- loop: If true, the animation will be played back n-times at original speed instead of being stretched according to the given duration.[8]

Children of child elements:

If <moveTo> has a <locomotion> child (whose 'refId' attribute references to a locomotion definition), the corresponding animations are taken for display along the calculated path. If it has a pathPos and/or pathOrient child with at least two path coordinates, these tags have precedence over the on-line calculated path given via the 'pos' and 'orientation' attributes.

Grammar:
```
ANIMATECHR ::= <animate id="ID" refId="ID" [speed="FLOAT"] ALIGNMENT>
              (<face refId="ID" intensity="FLOAT" />|<(gesture|posture) refId="ID" />)^{0,1}
                 POSEREF^{0,1}
              </animate>

POSEREF ::= <singlePose refId="ID" intensity="FLOAT" [loop="BOOL"] />
          | <implicitPose refId="ID" target="ID" [loop="BOOL"] />
          | <multiPoses refId="ID" [loop="BOOL"] />
          | <moveTo refId="ID" [pos="VECTOR"] [orientation="QUATERNION"] [loop="BOOL"]>

                LOCOMOTIONREF^{0,1}
                PATHPOS^{0,1}
                PATHROT^{0,1}
            </moveTo>

LOCOMOTIONREF ::= <locomotion refId="ID" />
PATHPOS ::= <pathPos>
                VECTOR*
            </pathPos>
PATHROT ::= <pathOrient>
                QUATERNION*
            </pathOrient>
```

3.1.6 Element: <stopAnimate>

Stops a running animation.

Attributes:

[8] PML 2.1

- id: Identifier of the action.
- refId : Name of the referenced animation.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
STOPANIMATE ::= <stopAnimate id="ID" refId="ID" ALIGNMENT />
```

3.1.7 Element: <complexion>

Determines the facial color of the character. After the specified time it changed into the default facial color.

Attributes:

- id: Identifier of the action.
- refId: Reference to a certain skin texture.
- dur: Duration of the facial coloration.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
COMPLEXIONACT ::= <complexion id="ID" refId="ID" dur="NATURAL"
                   intensity="FLOATINTERVAL" ALIGNMENT />
NATURAL       ::= 0|([1-9][0-9]*)
FLOATINTERVAL ::= (0.[0-9]+)|1.0
```

3.1.8 Element: <speak>

The synthesized speech output is controlled by the <speak> element. It is possible to expand the given text with specific elements to control the articulation of words and sentences.

Attributes:

- id: Identifier of the action.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

child elements:

- <text>: Specify the text that will be spoken
- <audio> (optionally): Sound file.

Grammar:

```
SPEAK ::= <speak id="ID" ALIGNMENT>
            SPEAKTEXT
            (SPEAKAUDIO)^{0,1}
          </speak>
```

3.1.9 Element: <text>

Specification of the spoken text. The pronunciation of the text can be provided with additional elements which are already known from SSML.

child elements:

- <break> (optionally): Break within the text.
- <prosody> (optionally): Modifies the pronunciation of the text.
- <emphasis> (optionally): Accentuated the text.

Attributes of the child elements:

Element: break

- time: Duration of the break (in ms).

Element: prosody

- pitch (optionally): Pitch of the voice.
- range (optionally): Area of the pitch of the voice.
- rate (optionally): Speed of the voice.
- volume (optionally): Sound intensity.

Element: emphasis

- level: Type of the accent.

Grammar:

```
SPEAKTEXT ::= <text>TEXT</text>
TEXT ::= TEXTTEXT
      | ALPHATEXT+
      | <break time="NATURAL" />
      | <prosody [pitch="PITCHLEVEL"] [range="PITCHLEVEL"] [rate="RATELEVEL"] [volume="VOLUMELEVEL"]>
             TEXT
        </prosody>
      | <emphasis level="EMPHASISLEVEL">TEXT</emphasis>

ALPHATEXT      ::= [a-zA-Z0-9,.;:!? -äöüß]
EMPHASISLEVEL ::= none|reduced|moderate|strong
PITCHLEVEL     ::= default|x-low|low|medium|high|x-high
RATELEVEL      ::= default|x-slow|slow|medium|fast|x-fast
VOLUMELEVEL    ::= silent|x-soft|soft|medium|loud|x-loud
```

Example:

```
<text>
  Hello<break time="50" />how<prosody rate="fast">are</prosody>you?
</text>
```

3.1.10 Element: <audio>

Audio files which were generated for specific texts and the according visemes can be specified with the <audio> element.

Attributes:

- src: Source of the audio file.

child elements:

- <phoneme> (optionally): Phonemes to play. The sequence of phonemes determines the play order.

Attributes of the child elements:

- refId: Reference to phoneme.
- dur: Play time of the phonemes (in ms).

Grammar:

```
SPEAKAUDIO   ::= <audio src="SRC">
                  (<phoneme refId="ID" dur="NATURAL" />)*
                 </audio>
SRC          ::= (file|http|rep)://SRCSEGMENT
SRCSEGMENT   ::= (SRCSEGMENT/SRCSEGMENT)|ALPHANUMPATH+
ALPHANUMPATH ::= [a-zA-Z0-9:.-_]
```

Example:

```
<audio src="rep://sven/hello.wav">
  <phoneme refId="h" dur="1000" />
  <phoneme refId="u" dur="500" />
  <phoneme refId="h" dur="1000" />
  <phoneme refId="u" dur="500" />
</audio>
```

3.1.11 Element: <pause>

This action simply does nothing for a given duration. Pause actions will only be used to control the generation of the schedule blocks. For this reason, pause actions will not be used inside a schedule block and are only of interest to PML generating modules.

Attributes:

- id: identifier of this action.
- dur: Duration of the pause action (in ms).
- alignTo: Specifies the action to which this one should be aligned.
- alignType: Specifies how this action should be aligned to another.

Grammar:

```
PAUSE ::= <pause id="ID" dur="NATURAL" ALIGNMENT />
```

3.2 Element: <object>

This element is used to specify all actions which were related to the referenced object (refID). It is possible to play or show multimedia data or show GUI elements on the specific object.
Note: Elements already defined in the previous section are not further explained here again.

Attributes:

- refId: Specifies the object which should perform an action.

child elements:

- <show>: Show object.
- <hide>: Hide object.
- <transform>: Position object.
- <startIdleList>, <stopIdleList>: enable idlePoses.
- <animate>: Animate object.
- <stopAnimate>: Stops animation.[9]
- <assignImage>, <startAudio>, <stopAudio>, <startVideo>, <stopVideo>: Show multimedia data.
- <assignText>: Show some text.[10]
- <assignMenu>: Show menu.
- <selectMenu>: Choose menu item.
- <assignSlider>: Show sliders.
- <pause>: Do nothing.
- <startFollowing>: Start following target.[11]
- <stopFollowing>: Stop following.[12]
- <zoom>: Zoom in or out.[13]
- <frameTarget>: Frame given targets.[14]

Grammar:

```
OBJECTACTION ::= <object refId="ID">
                   (SHOW|HIDE|TRANSFORM|IDLELIST|ANIMATEOBJ|
                   MEDIAACT|MENUACT|SLIDERACT|PAUSE|STOPANIMATE|
                   STARTFOLLOW|STOPFOLLOW|ZOOM|FRAMETARGET)+
                 </object>
```

[9]PML 2.1
[10]PML 2.1
[11]PML 2.1
[12]PML 2.1
[13]PML 2.1
[14]PML 2.1

3.2.1 Element: <animate>

Animates the object.

Attributes:

- id: Identifier of the action
- refId: Name of the referenced animation.
- speed (optionally): Animation speed (as a factor).
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

child elements:

- <singlePose> (optionally): Reference to singlePose.
- <multiPoses> (optionally): Reference to multiPoses.
- <implicitPose> (optionally): Reference to implicitPose.
- <moveTo> (optionally): Reference to fragment.[15]

Attributes of the child elements:

- refId: Name of the referenced animation.
- intensity (only singlePose): Intensity of the referenced singlePose.
- target (only implicitPose): The target of the referenced implicitPose.

Grammar:

```
ANIMATEOBJ ::= <animate id="ID" refId="ID" [speed="FLOAT"] ALIGNMENT>
                POSEOBJREF^{0,1}
               </animate>

POSEOBFREF ::= <singlePose refId="ID" intensity="FLOAT" [loop="BOOL"] />
             | <implicitPose refId="ID" target="ID" [loop="BOOL"] />
             | <multiPoses refId="ID" [loop="BOOL"] />
             | <moveTo refId="ID" [pos="VECTOR"] [orientation="QUATERNION"] [loop="BOOL"]>

                 PATHPOS^{0,1}
                 PATHROT^{0,1}
               </moveTo>
```

Example:

```
<animate id="animate1" refId="jump" alignTo="null" alignType="null">
  <multiPoses refId="move04" />
</animate>
```

[15] PML 2.1

3.2.2 Elements: <assignImage>, <assignText>, <startAudio>, <stopAudio>, <startVideo>, <stopVideo>

Show/play or pause multimedia content on the object.

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate menu.
- loop: Indicates if a sound or video file should be looped.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
MEDIAACT ::= <(startAudio|startVideo) id="ID" refId="ID" loop="BOOL" ALIGNMENT /> |
             <(assignImage|stopAudio|stopVideo) id="ID" refId="ID" ALIGNMENT /> |

             <assignText id="ID" refId="ID" ALIGNMENT >
               <text>ALPHATEXT+</text>
             </assignText>
```

3.2.3 Element: <assignMenu>

The <assignMenu> element can be used to show a menu on top of the specified object.

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate menu.
- title (optionally):Title of the menu.
- enabled: Enable or disable the GUI element.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

child elements:

- <entry> (optionally): menu item.

Attributes of the child elements:

- id: Identifier for the menu item.
- value: Return value when selecting this menu entry.

Grammar:

```
MENUACT ::= <assignMenu id="ID" refId="ID" [title="ALPHATEXT+"] enabled="BOOL" ALIGNMENT>
              (<entry id="ID" value="ID">ALPHATEXT+</entry>)*
            </assignMenu>
```

Example:

```
<assignMenu id="a124" refId="m0" title="Welche Farbe hat Strom?" enabled="true"
 alignTo="null" alignType="null">
  <entry id="A" value="false">Gelb</entry>
  <entry id="B" value="false">Blau</entry>
  <entry id="C" value="true">Keine</entry>
</assignMenu>
```

3.2.4 Element: <selectMenu>

With this element it is possible to choose one or more menu items.

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate menu.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Child elements:

- <entry>: menu item.

Attributes of the child elements:

- refId: Reference to the corresponding menu item.
- checked (optionally): Indicates if the menu item is selected or not (default: true).

Grammar:

```
MENUSELACT ::= <selectMenu id="ID" refId="ID" ALIGNMENT>
                 (<entry refId="ID" [checked="BOOL"] />)+
               </selectMenu>
```

Example:

```
<selectMenu id="a1" refId="m0" alignTo="null" alignType="null">
  <entry refId="A"/>
</selectMenu>
```

3.2.5 Element: <assignSlider>

Show a slider on an object.

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate slider.
- value (optionally): Current value of the slider.
- enabled: Enable or disable this GUI element.
- alignTo: Indicates the action to which this action should be aligned temporal.
- alignType: Determines in which manner this action should be aligned.

Grammar:

```
SLIDERACT ::= <assignSlider id="ID" refId="ID" [value="INTEGER"] enabled="BOOL" ALIGNMENT />
```

3.2.6 Element: <startFollowing>

Start following a target (only valid for camera).

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate camera.
- mode: Optional follow mode.

Grammar:

```
STARTFOLLOW ::= <startFollowing id="ID" refId="ID" [mode="FOLMODE"] ALIGNMENT />

FOLMODE ::= pos|orient|posOrient
```

3.2.7 Element: <stopFollowing>

Stop following the target (only valid for camera).

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate camera.

Grammar:

```
STOPFOLLOW ::= <stopFollowing id="ID" refId="ID" ALIGNMENT />
```

3.2.8 Element: <activateEffect>

Activates a certain visual effect, e.g. sketchy rendering (only valid for camera).

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate camera.
- type: Allows to define special camera dependent visual effects.

Grammar:

```
ACTEFFECT ::= <activateEffect id="ID" refId="ID" type="FXTYPE" ALIGNMENT />

FXTYPE ::= depthOfField|blur|motionBlur|glow|ambientOcclusion|sketch
```

3.2.9 Element: <deactivateEffect>

Deactivates a certain visual effect, e.g. sketchy rendering (only valid for camera).

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate camera.
- type: A special camera dependent visual effect.

Grammar:

```
DEACTEFFECT ::= <deactivateEffect id="ID" refId="ID" type="FXTYPE" ALIGNMENT />
```

3.2.10 Element: <zoom>

Zoom in or out (only valid for camera).

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate camera.
- fov: The field of view, which gets interpolated from actual value to 'fov' during 'dur'.

Grammar:

```
ZOOM ::= <zoom id="ID" refId="ID" fov="FLOAT" ALIGNMENT />
```

3.2.11 Element: <frameTarget>

Frame given targets (only valid for camera).

Attributes:

- id: Identifier of the action.
- refId: Reference to the appropriate camera.
- shotSize (optionally): The shot type, e.g. "full" or "close-up".
- angle (optionally): Offset angle from line-of-action (yaw).
- pitch (optionally): Offset angle to horizontal plane.
- roll (optionally): Offset angle to vertical plane (unusual).

Child elements:

- <minScreenPos>: The minimum screen position in normalized coordinates (in $[0;1]$). The number of entries must correspond to the number of <targetFull> tags as defined in <camera>.
- <maxScreenPos>: The maximum screen position in normalized coordinates (in $[0;1]$). The number of entries must correspond to the number of <targetCloseUp> tags as defined in <camera>.

Grammar:

```
FRAMETARGET ::= <frameTarget id="ID" refId="ID" [shotSize="SHOTTYPE"]
                 [angle="FLOAT"] [pitch="FLOAT"] [roll="FLOAT"] ALIGNMENT>
                   MINSCREEN^{0,1}
                   MAXSCREEN^{0,1}
                </frameTarget>
MINSCREEN   ::= <minScreenPos>
                   VECTOR2^+
                </minScreenPos>
MAXSCREEN   ::= <maxScreenPos>
                   VECTOR2^+
                </maxScreenPos>

VECTOR2      ::= FLOAT,FLOAT
SHOTTYPE     ::= extremeLong|long|full|mediumFull|medium|mediumClose|
                 close|wideCloseup|closeup|mediumCloseup
```

3.3 Element: <schedule>

The <schedule> element describes the temporal sequence of actions within a schedule-block. Actions, which should be executed sequentially, are embedded in an *seq* element. Parallel actions are embedded in an *par* element.

The task of the schedule-block, which can be generated by a special ActionEncoder or any other higher level PML generating module, is to tell the player/ visualization engine, when which action shall be executed. As already mentioned in section 3.1.5, on the player side all timing information is taken from this <schedule>!

child elements:

- <action>: Tasks to be done.
- <par>: Perform actions parallel.
- <seq>: Perform actions sequentially.

Attributes of the child elements:

- refId (only action): Reference to the action.
- begin (only action): Start point of the action (in ms).
- dur (only action): Duration in ms.

Grammar:

```
SCHEDULE    ::= <schedule>
                  SUBSCHEDULE
                </schedule>
SUBSCHEDULE ::= ACTION
             | <par>
                  SUBSCHEDULE
                  SUBSCHEDULE+
               </par>
             | <seq>
                  SUBSCHEDULE
                  SUBSCHEDULE+
               </seq>
ACTION      ::= <action refId="ID" begin="NATURAL" dur="NATURAL" />
```

Example:

```
<schedule>
  <par>
    <action refId="breath" begin="0" dur="10000" />
    <seq>
      <action refId="think" begin="0" dur="4500" />
      <action refId="eat" begin="0" dur="4500" />
    </seq>
  </par>
</schedule>
```

4 Top level element: <message>

The <message> element is used to exchange messages within the system. There are three different types of messages. *Commands* are used to control the player, *States* reports the success or failure of commands and *Facts* are used to inform the system about user interaction (for example user has decided to answer b).

Attributes:

- id: Identifiers.

child elements:

- <command>: Commands to the player.
- <state>: Status reports on the player.
- <fact>: Different kind of informations.
- <userData>: Optional user specific data such as meta data or ontology information (given as XML subtree), which is not evaluated by the player but passed through.

Attributes of child elements:

Elements: command, state

- refId: Name of the script for reference. In case of startScreenDump/stopScreenDump refId can refer to the viewport/viewarea that shall be dumped, and in case of reset, the value of refId is not relevant as the whole PML handler is set back.
- type: Type of commands.
- description (only state): Description (for example an error).

Element: fact

- refId: Reference to a scene object.
- value: Value of the attribute.

Element: userData

- description: Can be used for describing the type of user data (optional).

The <userData> child element can be used to pass through data (that is irrelevant for the player) from higher level modules to other backends, which require additional information beyond what is provided by PML. It is especially of interest in combination with the 'startScreenDump' and 'stopScreenDump' messages. If these are triggered, besides dumping frames an additional footage descriptor file is written in XML format, which can be enriched with the information that is given as a child node of the <userData> element.

Grammar:

```
MESSAGE ::= <message id="ID">
              (<command refId="ID" type="COMMANDTYPE" />|
               <state refId="ID" type="STATETYPE" [description="ALPHATEXT+"] />|
               <fact refId="ALPHATEXT+" value="ALPHATEXT+" />)
              USERDATA 0,1
            </message>

COMMANDTYPE ::= reset|start|stop|
                startScreenDump|stopScreenDump| startPauseAnimation|stopPauseAnimation[16]
```

[16]The latter four command types are part of the PML 2.1 draft.

```
STATETYPE   ::= failed|fetched|started|stopped|finished|undefined

ALPHATEXT   ::= [a-zA-Z0-9,.;:!? -äöüß]

ID          ::= [a-zA-Z0-9.:_-]+

USERDATA    ::= <userData [description="ALPHATEXT+"] >
                  XML
                </userData>
```

In this context, `XML` denotes unspecified XML user data in the sense of a CDATA section.

Examples:

```
<message id="c_042">
  <command refId="script076" type="start" />
</message>

<message id="s_043">
  <state refId="script076" type="failed" description="Not enough memory" />
</message>

<message id="f_172">
  <fact refId="m0" value="01" />
</message>

<message id="msg000">
  <command refId="avalon" type="reset"/>
</message>
```

5 Top level element: <query>

Queries to the player are not yet specified.

Grammar:

```
QUERY ::=
```

6 Grammar

The whole PML 2.1 grammar is defined in Backus-Naur form.
Note: please refer to chapters 1, 2, 3, and 4 for the complete PML specification including description and examples.

6.1 PML2 document

```
PML2DOC ::= DEFINITIONS|ACTIONS|MESSAGE|QUERY
```

6.1.1 Definitions

```
DEFINITIONS ::= <definitions id="ID">
                 REPOSITORY*
                 UNDEFINE*
                 (CHARACTER|OBJECT)*
                </definitions>

REPOSITORY ::= <repository id="ID">
                PATH+
               </repository>

PATH ::= <path src="SRCSIMPLE" />

UNDEFINE ::= <undefine refId="ID" />

CHARACTER ::= <character id="ID" src="SRC" [root="ID"]>
                VOICE 0,1
                VISEME 0,1
                SOUNDSOURCE 0,1
                (CANVAS|FRAGMENT|TARGET|COMPLEXION|SINGLEPOSE|MULTIPOSES|
                 IMPLICITPOSE|CREATESINGLEPOSE|IDLEPOSES|CREATEMULTIPOSES|LOCOMOTION)*
              </character>

VOICE       ::= <voice id="ID" refId="ID" pitch="PITCHLEVEL"
                 range="PITCHLEVEL" rate="RATELEVEL" volume="VOLUMELEVEL" />

VISEME ::= <viseme>
             PHONEME+
           </viseme>

PHONEME ::= <phoneme id="ID" refId="ID" intensity="FLOAT" />

SOUNDSOURCE ::= <soundSource id="ID" [src="SRC"] [root="ID"]/>

CANVAS ::= <canvas id="ID" [src="SRC"] [root="ID"] />

FRAGMENT ::= <fragment id="ID" [src="SRC"] [root="ID"] />

TARGET ::= <target id="ID" root="ID" />

COMPLEXION ::= <complexion id="ID" refId="ID" src="SRC" />

SINGLEPOSE ::= <singlePose id="ID" src="SRC" [root="ID"] [dur="NATURAL"] />
```

```
IMPLICITPOSE ::= <implicitPose id="ID" src="SRC" type="IKTYPE" [dur="NATURAL"] />

MULTIPOSES    ::= <multiPoses id="ID" src="SRC" [root="ID"] [from="FLOATINTERVAL"]
                   [to="FLOATINTERVAL"] [dur="NATURAL"] />

CREATESINGLEPOSE ::= <createSinglePose id="ID">
                       SINGLEPOSEREF+
                     </createSinglePose>

SINGLEPOSEREF    ::= <singlePose refId="ID" intensity="FLOAT" />

CREATEMULTIPOSES ::= <createMultiPoses id="ID" [dur="NATURAL"]>
                       MULTIPOSESREF+
                     </createMultiPoses>

MULTIPOSESREF    ::= <multiPoses refId="ID" weight="FLOAT" />

LOCOMOTION ::= <locomotion id="ID" [dur="NATURAL"]>
                 <speed minSpeed="FLOAT" maxSpeed="FLOAT">
                    MULTIPOSESREF+
                 </speed>
                 <turnAngle maxLeft="FLOAT" maxRight="FLOAT">
                    MULTIPOSESREF+
                 </turnAngle>
               </locomotion>

IDLEPOSES ::= <idlePoses id="ID" random="BOOL">
                POSEDURREF+
              </idlePoses>

POSEDURREF ::= <singlePose refId="ID" intensity="FLOAT" dur="NATURAL" />
             | <implicitPose refId="ID" target="ID" [dur="NATURAL"] />
             | <multiPoses refId="ID" [dur="NATURAL"] />

OBJECT ::= <object id="ID" src="SRC" [root="ID"]>
             (CANVAS|FRAGMENT|GUICONTAINER|SOUNDSOURCE|TARGET|SINGLEPOSE|IMPLICITPOSE|
              MULTIPOSES|CREATESINGLEPOSE|IDLEPOSES|MEDIA|MENU|SLIDER|CAMERA)*
           </object>

GUICONTAINER      ::= <guiContainer id="ID" [src="SRC"] [root="ID"]
                       [device="ID"] visualization="VISUALIZATIONTYPE" />

MEDIA ::= <(audio|image|video) id="ID" refId="ID" src="SRC"/>

MENU ::= <menu id="ID" refId="ID" multi="BOOL" src="SRC" />

SLIDER ::= <slider id="ID" refId="ID" min="INTEGER" max="INTEGER" src="SRC" />

CAMERA ::= <camera id="ID" [facingDir="VECTOR"] [fov="FLOAT"] [src="SRC"] [root="ID"] >
             TARGETFULL+
             TARGETCLOSE+
           </camera>
```

```
TARGETFULL ::= <targetFull id="ID" root="ID" />

TARGETCLOSE ::= <targetCloseUp id="ID" root="ID" />
```

6.1.2 Actions

```
ACTIONS ::= <actions id="ID" start="BOOL">
              (CHARACTERACTION|OBJECTACTION)^*
              SCHEDULE ^{0,1}
            </actions>

CHARACTERACTION ::= <character refId="ID">
                      (SHOW|HIDE|TRANSFORM|IDLELIST|ANIMATECHR|COMPLEXIONACT|SPEAK|PAUSE|STOPANIMATE)^+
                    </character>
OBJECTACTION ::= <object refId="ID">
                   (SHOW|HIDE|TRANSFORM|IDLELIST|ANIMATEOBJ|
                   MEDIAACT|MENUACT|SLIDERACT|PAUSE|STOPANIMATE|
                   STARTFOLLOW|STOPFOLLOW|ZOOM|FRAMETARGET)^+
                 </object>

SHOW      ::= <show id="ID" refId="ID" ALIGNMENT />

HIDE      ::= <hide id="ID" refId="ID" ALIGNMENT />

TRANSFORM  ::= <transform id="ID" refId="ID" [pos="VECTOR"]
               [orientation="QUATERNION"] [scale="VECTOR"] ALIGNMENT />

IDLELIST ::= <startIdleList id="ID" refId="ID" [minDelay="NATURAL"]
              [maxDelay="NATURAL"] [concurrent="BOOL"] ALIGNMENT /> |
             <stopIdleList id="ID" refId="ID" ALIGNMENT />

ANIMATECHR ::= <animate id="ID" refId="ID" [speed="FLOAT"] ALIGNMENT>
               (<face refId="ID" intensity="FLOAT" />|<(gesture|posture) refId="ID" />)^{0,1}
                 POSEREF^{0,1}
               </animate>

POSEREF ::= <singlePose refId="ID" intensity="FLOAT" [loop="BOOL"] />
          | <implicitPose refId="ID" target="ID" [loop="BOOL"] />
          | <multiPoses refId="ID" [loop="BOOL"] />
          | <moveTo refId="ID" [pos="VECTOR"] [orientation="QUATERNION"] [loop="BOOL"]>

                LOCOMOTIONREF^{0,1}
                PATHPOS^{0,1}
                PATHROT^{0,1}
            </moveTo>

LOCOMOTIONREF ::= <locomotion refId="ID" />

PATHPOS ::= <pathPos>
                VECTOR^*
            </pathPos>

PATHROT ::= <pathOrient>
                QUATERNION^*
```

```
            </pathOrient>

STOPANIMATE ::= <stopAnimate id="ID" refId="ID" ALIGNMENT />

COMPLEXIONACT ::= <complexion id="ID" refId="ID" dur="NATURAL"
                   intensity="FLOATINTERVAL" ALIGNMENT />

SPEAK ::= <speak id="ID" ALIGNMENT>
            SPEAKTEXT
            (SPEAKAUDIO)^{0,1}
          </speak>

SPEAKTEXT ::= <text>TEXT</text>

TEXT ::= TEXTTEXT
      | ALPHATEXT^+
      | <break time="NATURAL" />
      | <prosody [pitch="PITCHLEVEL"] [range="PITCHLEVEL"] [rate="RATELEVEL"] [volume="VOLUMELEVEL"]>
             TEXT
        </prosody>
      | <emphasis level="EMPHASISLEVEL">TEXT</emphasis>

SPEAKAUDIO   ::= <audio src="SRC">
                  (<phoneme refId="ID" dur="NATURAL" />)^*
                 </audio>)

ANIMATEOBJ ::= <animate id="ID" refId="ID" [speed="FLOAT"] ALIGNMENT>
                POSEOBJREF^{0,1}
               </animate>

POSEOBFREF ::= <singlePose refId="ID" intensity="FLOAT" [loop="BOOL"] />
             | <implicitPose refId="ID" target="ID" [loop="BOOL"] />
             | <multiPoses refId="ID" [loop="BOOL"] />
             | <moveTo refId="ID" [pos="VECTOR"] [orientation="QUATERNION"] [loop="BOOL"]>
                 PATHPOS^{0,1}
                 PATHROT^{0,1}
               </moveTo>

MEDIAACT ::= <(startAudio|startVideo) id="ID" refId="ID" loop="BOOL" ALIGNMENT /> |
             <(assignImage|stopAudio|stopVideo) id="ID" refId="ID" ALIGNMENT /> |
             <assignText id="ID" refId="ID" ALIGNMENT >
               <text>ALPHATEXT^+</text>
             </assignText>
MENUACT ::= <assignMenu id="ID" refId="ID" [title="ALPHATEXT^+"] enabled="BOOL" ALIGNMENT>
             (<entry id="ID" value="ID">ALPHATEXT^+</entry>)^*
            </assignMenu>

MENUSELACT ::= <selectMenu id="ID" refId="ID" ALIGNMENT>
                 (<entry refId="ID" [checked="BOOL"] />)^+
               </selectMenu>
```

SLIDERACT ::= <assignSlider id="ID" refId="ID" [value="INTEGER"] enabled="BOOL" ALIGNMENT />

PAUSE ::= <pause id="ID" dur="NATURAL" ALIGNMENT />

STARTFOLLOW ::= <startFollowing id="ID" refId="ID" [mode="FOLMODE"] ALIGNMENT />

STOPFOLLOW ::= <stopFollowing id="ID" refId="ID" ALIGNMENT />

ACTEFFECT ::= <activateEffect id="ID" refId="ID" type="FXTYPE" ALIGNMENT />

DEACTEFFECT ::= <deactivateEffect id="ID" refId="ID" type="FXTYPE" ALIGNMENT />

ZOOM ::= <zoom id="ID" refId="ID" fov="FLOAT" ALIGNMENT />

FRAMETARGET ::= <frameTarget id="ID" refId="ID" [shotSize="SHOTTYPE"]
[angle="FLOAT"] [pitch="FLOAT"] [roll="FLOAT"] ALIGNMENT>
MINSCREEN^{0,1}
MAXSCREEN^{0,1}
</frameTarget>
MINSCREEN ::= <minScreenPos>
VECTOR2^{+}
</minScreenPos>
MAXSCREEN ::= <maxScreenPos>
VECTOR2^{+}
</maxScreenPos>

SCHEDULE ::= <schedule>
SUBSCHEDULE
</schedule>
SUBSCHEDULE ::= ACTION
| <par>
SUBSCHEDULE
SUBSCHEDULE^{+}
</par>
| <seq>
SUBSCHEDULE
SUBSCHEDULE^{+}
</seq>
ACTION ::= <action refId="ID" begin="NATURAL" dur="NATURAL" />

6.1.3 Messages

MESSAGE ::= <message id="ID">
(<command refId="ID" type="COMMANDTYPE" />|
<state refId="ID" type="STATETYPE" [description="ALPHATEXT^{+}"] />|
<fact refId="ALPHATEXT^{+}" value="ALPHATEXT^{+}" />)
USERDATA ^{0,1}
</message>

```
USERDATA ::= <userData type= [description="ALPHATEXT+"] >
               XML
             </userData>
```

```
XML ::= unspecified user data
```

6.1.4 Queries

```
QUERY ::=
```

6.2 Basic grammar

```
ALPHA ::= [a-zA-Z]

ALPHANUM ::= [a-zA-Z0-9]

ALPHANUMPATH ::= [a-zA-Z0-9:.-_]

ALPHATEXT ::= [a-zA-Z0-9,.;:!? -äöüß]

BOOL ::= true|false

NATURAL ::= 0|([1-9][0-9]*)

INTEGER ::= [-]NATURAL

FLOAT ::= 0|[-]((0.[0-9]+)|([1-9][0-9]*[.[0-9]+]))

FLOATINTERVAL ::= (0.[0-9]+)|1.0

ID ::= [a-zA-Z0-9.:_-]+

ALIGNMENT ::= [alignTo="null|ID"] [alignType="ALIGNTYPE"]

ALIGNTYPE ::= null|equals|before|after|meets|met-by|overlaps|
              overlapped-by|starts|started-by|during|contains|
              finishes|finished-by|starts-with|finishes-with

SRC ::= (file|http|rep)://SRCSEGMENT

SRCSIMPLE ::= (file|http)://SRCSEGMENT

SRCSEGMENT ::= (SRCSEGMENT/SRCSEGMENT)|ALPHANUMPATH+

VECTOR  ::= FLOAT,FLOAT,FLOAT

VECTOR2 ::= FLOAT,FLOAT

QUATERNION ::= VECTOR,FLOAT[17]
```

[17]Note: actually the QUATERNION token stands for Axis-Angle representation.

```
COMMANDTYPE ::= reset|start|stop|
                startScreenDump|stopScreenDump|
                startPauseAnimation|stopPauseAnimation

FOLMODE ::= pos|orient|posOrient

FXTYPE  ::= depthOfField|blur|motionBlur|glow|ambientOcclusion|sketch

IKTYPE ::= lookAt|lookAtHold|lookAtRetract|eyeGazeAt|eyeGazeAtHold|eyeGazeAtRetract|
           pointLeftHand|pointLeftHandHold|pointLeftHandRetract|
           pointRightHand|pointRightHandHold|pointRightHandRetract

SHOTTYPE ::= extremeLong|long|full|mediumFull|medium|mediumClose|
             close|wideCloseup|closeup|mediumCloseup

STATETYPE ::= failed|fetched|started|stopped|finished|undefined

VISUALIZATIONTYPE ::= 2D|3D

EMPHASISLEVEL ::= none|reduced|moderate|strong

PITCHLEVEL ::= default|x-low|low|medium|high|x-high

RATELEVEL ::= default|x-slow|slow|medium|fast|x-fast

VOLUMELEVEL ::= silent|x-soft|soft|medium|loud|x-loud
```

7 Annex – Controlling X3D with PML

7.1 Notes

Because PML follows an approach of iterative refinement, some tags and attributes are only useful at certain points in the pipeline. Whereas higher level modules do their scheduling based on the **ALIGNMENT** property, this property is ignored on the lower rendering and animation level. The same goes e.g. for the *face*, *gesture*, and *posture* tags, which are replaced on their way through the module pipe by the more concrete tags *singlePose*, *multiPoses*, and *implicitPose*.

This is due to the fact, that absolute scheduling can only be done after all information is present, such as the timings of the text-to-speech (TTS) system or the duration of a given character animation. So, in this context PML can either be used as an interface language for different modules, which propagates information that gets enriched with each module in a typical virtual human module pipeline, or it can directly be used for scripting behavior. But in both cases, the visualization engine only uses the concrete animation information given by the schedule block.

If you want to learn more about the development of PML, its backgrounds and possible use cases, you should also have a look at the references section.

Implementation Details

Note: This section is NOT part of the spec. Currently the "voice" tag, the new pause messages and the "assignText" tag as well as some of the 'root' attributes are ignored in Instant Player. Besides this, internal locomotion handling of "moveTo" is not yet implemented. Moreover, all extensions of the PML 2.1 draft (marked by a footnote on first definition) may still change...
Every Character and Object definition in PML requires an "AnimationController" node in the corresponding X3D setup files (the mapping is achieved via the 'id' attribute in PML and the 'name' field in X3D respectively). The same goes for almost every definition of a specific animation or transition, which maps to an "AnimationContainer" node. Whereas in many cases the "AnimationController" can be created at runtime when defining a new character or object via PML (if you don't need special settings), this is not possible for the "AnimationContainer" nodes, which are referenced by child elements like *fragment* or *multiPoses*, because PML is by design a language for controlling behavior – but not for creating it.

7.2 Introducing an Animation Control Language

This chapter coarsely covers how to control X3D[18] scenes containing H-Anim characters and other objects with PML (Figure 1 visualizes the basic idea). Therefore, Instant Reality's[19] new BehaviorController component is shortly introduced, which holds nodes that can be used for high-level animation and behavior control including synchronization by means of a special interface and description language building on top of X3D – namely PML (Player Markup Language).
Similar to the additional programming languages needed for X3D "Script" and "Shader" nodes[20], a domain specific language is introduced for animations. Besides, when animating and visualizing virtual characters one also has to think about interoperability aspects. Thus, especially in web environments, it should be possible to specify the properties and behaviors of characters and objects in a virtual environment independently from their realization in a concrete setting, whilst still being able to provide detailed information like the required animation parameters and exact timing information. Therefore the Player Markup Language (PML) was developed.
PML is an XML-based high level markup language for scripting the behavior controller component[21], and thus comparable to a "Script" or "Shader" node, as it is a domain specific language to extend the current X3D concept.

[18] http://www.web3d.org/
[19] http://www.instantreality.org/
[20] http://www.web3d.org/x3d/specifications/
[21] http://www.instantreality.org/documentation/component/BehaviorController/

Cognition (AI)	
(Instinctive) Behavior	PML
Animations (Joint & Vertex Transformations)	X3D
Shape (Skeleton, Geometry, Appearance)	

Figure 1: Layers of complexity, with basic technology used.

Additionally, it is designed to be independent of the implementation of a graphics engine and virtual environment, and hence can either be used as descriptive interface markup language between a graphics engine and some higher level behavior and dialog generation engines, or for directly scripting animations. Because PML is a language for controlling virtual environments with special regards to character animation, it defines a format for sending appropriate commands. At the beginning of a new scene all objects and characters are defined by a `<definitions>` script. There exist three types of definitions: repository definitions, character definitions, and object definitions. Repository definitions specify where the resources for the various scene elements are located. Character definitions specify the acoustic parameters of the synthetic voice, the available animations including their default durations, the phoneme to viseme mapping to be used, etc. (a short example that defines a list of idle animations is shown next). Likewise object definitions are used to specify cameras, user interface elements, and various media types that will be used in the scenario. Each such element has a unique 'id' by which it can be referenced via the 'refId' attribute in other elements.

```
<definitions id="iListDef">
  <character id="Valerie" src="Valerie.wrl">
    <multiPoses id="a" src="a.wrl" dur="2350"/>
    <multiPoses id="b" src="b.wrl" dur="2533"/>
    <idlePoses id="iP" random="true">
      <multiPoses refId="a" dur="2350"/>
      <multiPoses refId="b" dur="2533"/>
    </idlePoses>
  </character>
</definitions>
```

7.3 Handling Animations with PML

In the course of the story all runtime dependent actions like character animations are described by so-called `<actions>` scripts, whereas the temporal order is given by a special scheduling block including sequential and parallel elements. Actions are used to specify the appearance and behavior of all characters and objects in the environment. Some actions like 'show', 'hide', 'transform', or 'startIdleList' can be applied to both, characters and objects, while others are only available for specific scene elements. Below a short example script is shown, in which the previously defined idle list is started. Examples of actions that are only available for virtual characters are 'speak' for verbal output, and 'complexion' for the change in skin color like blushing and pallor.
PML defines a message format, which can be sent to the animation control component or received from it for enabling interactions with the scene via `<message>` scripts. A PML message is used to control the execution of actions and to exchange information between modules. There are three different types of messages: commands, states, and facts. Commands can be used to trigger the execution of actions; states are used to inform other modules about the execution state, e.g. started, failed, finished, what is important for later synchronization; and facts, which are represented by attribute-value pairs, can be used to inform about user actions.

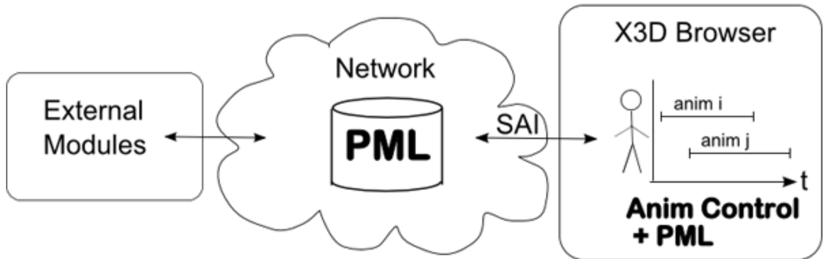

Figure 2: A rare but essential additional use-case: Utilizing SAI to send PML chunks during runtime to the animation controller.

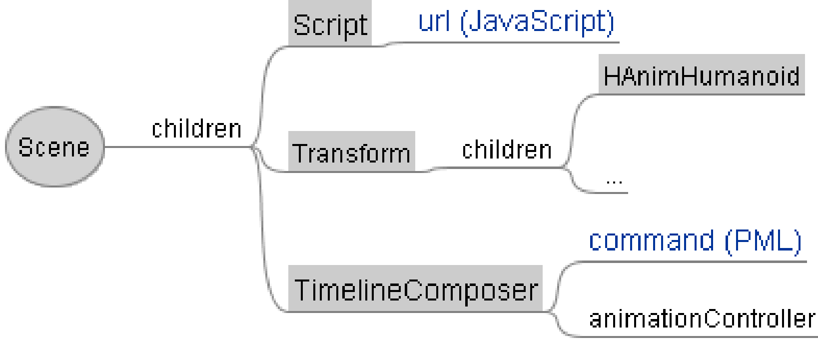

Figure 3: Usage of the animation scripting language PML within X3D.

```
<actions id="iListStart" start="true">
  <character refId="Valerie">
    <startIdleList id="iL" refId="iP" />
  </character>
  <schedule>
    <action refId="iL" begin="0" dur="0"/>
  </schedule>
</actions>
```

The animation tags of a PML actions script can either refer to preloaded animations, which are referenced by their name, or to simulated animations, e.g. via inverse kinematics. Different kinds of animations like morph targets and displacers for facial animation (`<singlePose>`), or key-frame animations (`<multiPoses>`) and simulated animations (`<implicitPose>`) for gestures and postures are distinguished, because every animation type must be handled differently and has a varying set of attributes. An example of a rather unusual animation which can be handled quite easily this way, too, is the change of the face complexion. Usually only the changes in geometry by means of displacers or morph targets are addressed in computer graphics. This is a well known problem, and the classification usually is based on the so-called Facial Action Coding System (FACS), which identifies certain Action Units for morphing the face geometry. But with the help of modern graphics hardware the more subtle changes concerning face coloring can also be covered via animated skin textures or shader programs.
By introducing a more abstract mechanism to define and synchronize different kinds of animations without having to take care about correct routing, timing etc., it is also much easier to create digital stories with embodied conversational agents in X3D. Such a story can be described with PML by putting together story-lines, i.e. short scene acts, in an easy and intuitive way through PML scripts that define when and what a character or object in the scene is doing. By combining this with other script or sensor nodes that define when and how the user can interact with the virtual environment there can also be added some non-linearity and possibilities for user interaction in order to create an interesting story graph. Figures 2 and 3 show a possible system setup. As can be seen, PML can either be used for scripting and synchronizing within the X3D browser, or for handling the communication with modules that do not want to bother with problems concerning low level kinematics.

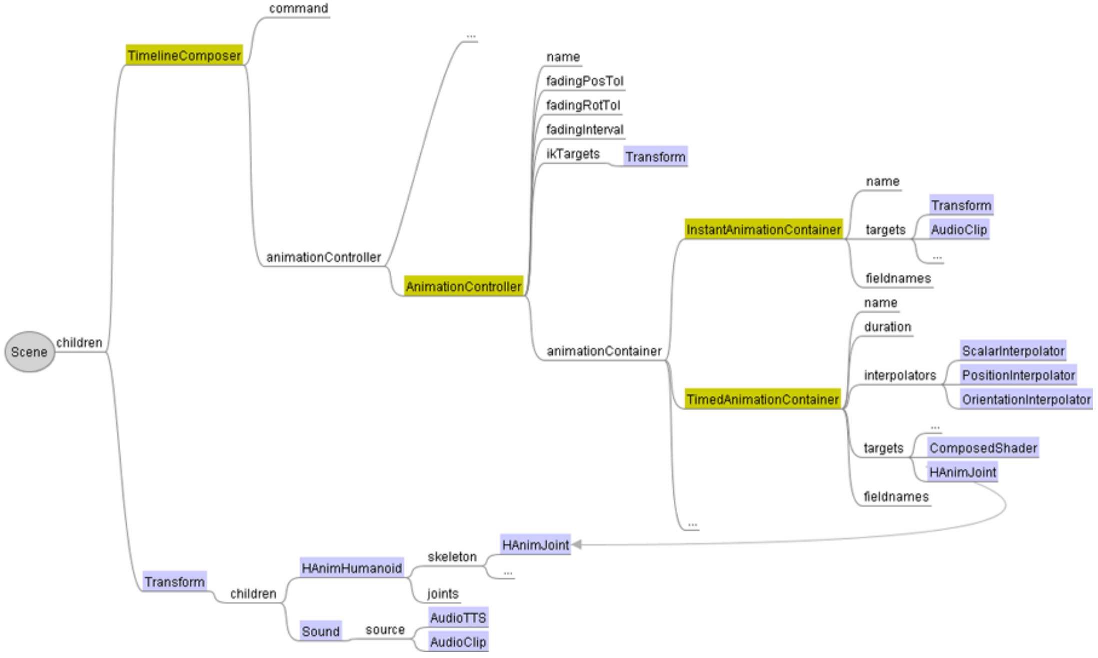

Figure 4: Overview of the BehaviorController component and its relations to other X3D nodes. Whenever the TimelineComposer receives a PML command, all requests are processed, and forwarded to the responsible AnimationController nodes.

7.4 Scheduling and Controlling Animations

7.4.1 Timeline and PML-Processor

After having explained the high level language PML, the question remains, how this advanced animation control approach can be used in concrete settings. Thus, a generic scheduling and controlling element is needed, too (Figure 2 gives a first impression on how to use it).

Figure 4 shows an overview of the proposed system architecture. Here, the TimelineComposer node is responsible for all scheduling and also deals as the PML interface and processor. Starting and stopping of animations and other actions is accomplished by setting the 'command' string of the TimelineComposer node with a valid PML file or string for defining the desired temporal order. This is similar in spirit to the 'url' field of a Shader node, which only is useful when having defined a valid GLSL or Cg shader program, or the 'url' field of a Script node, which only is useful when having defined some Java or JavaScript code.

Whereas the 'command' field contains an incoming PML script, the 'message' eventOut sends an outgoing PML message string. This way, the TimelineComposer node handles all communication with the system and forwards PML commands to its parser. During parsing, the scheduling block is sequenced and single action and definition chunks are created and transfered to the appropriate components. When having received a start message, the internal scheduler dispatches the action chunks to the AnimationController node of the corresponding character. The MFNode field 'animationController' holds references to the AnimationController nodes of all objects, which shall be animated. Whenever an actions script shall be executed, the TimelineComposer triggers all AnimationController nodes, which in turn access the respective data of their referenced animation container nodes (the InstantAnimationContainer for referring to transitions, which are state changes like toggling visibility, and the TimedAnimationContainer for storing all time based animations like key-frame animations) for processing this request.

7.4.2 AnimationController and AnimationContainer

The AnimationController node controls a set of animations connected with a virtual character or any other object. Because a complex application can lead to an arbitrary number of postures and gestures or respectively animations, the main job of the AnimationController is to blend and cross-fade all kinds of animations. This is due to the requirement, that for correct blending, cross-fading, and generally updating the actions of an object at a single time step, the controlling unit needs knowledge of all animations, a task that sometimes can not be handled with the simple scripting

and routing mechanisms of X3D. The 'name' field contains the name of the object to be controlled, which is relevant for the later mapping to PML scripts: X3D 'name' and PML 'id' must correspond for enabling the mapping! The 'animationContainer' field contains references to all animations as defined by the animation container nodes.
AnimationContainer nodes contain the animated targets of an animation. The MFString field 'targetnames' references the target nodes to be animated or changed (usually the joints), and the MFString field 'fieldnames' contains the names of the corresponding fields in order to find this field inside the target. This is needed, because if for instance an SFVec3f value shall be sent to a target node, e.g. a Transform node, it is often ambiguous, which field was meant (in this example it could be either of 'center', 'scale', or 'translation').
The TimedAnimationContainer node contains a set of interpolators of an animation (in the 'interpolators' MFNode field) and the original default duration of the animation (in the 'duration' field). Whereas the TimedAnimationContainer denotes actions with a certain duration, the InstantAnimationContainer denotes transitions, i.e. simple state changes like show, hide, start or stop, which have a duration of 0. It therefore does not contain interpolators but instead it can hold the id of a media-object as defined in a definitions script.

```
DEF AC_Valerie AnimationController {
  name "Valerie"
  fadingInterval 0
  animationContainer [
    DEF attract TimedAnimationContainer {
      name "attract"
      interpolators [
        OrientationInterpolator {
            #...
        },
        #...
      ]
      duration 4.2
      targetnames [
        "Bip01_Spine",
        #...
      ]
      fieldnames [
        "rotation",
        #...
      ]
    },
    #...
  ]
}
```

After having explained the high level interface, it will be shown how this corresponds to the nodes contained in the behavior controller component, and how they can be used in a concrete setting. The code fragment above shows exemplarily how to define interpolator based animations in TimedAnimationContainer nodes, and how an AnimationController node for a character or object can look like. If you send a definitions script defining a new object, which not yet has a Controller node, then an AnimationController will be created automatically. If you also like the character or objects to be dynamically loaded on definition, you simply have to provide the correct file name and root node in the 'src' and 'root' attributes of the definition.
As mentioned, in this framework, the interpolators are only used as data containers for key-value pairs, as depicted in Figure 5. Thus, there is no need for routes or other difficult to maintain helper structures, because all interpolators, which are active at a given time frame t, are solely used for the internal calculation of joint rotations etc., in order to have access to all required animation data for mixing all animations, which are active at a given time t within the timeline, efficiently.

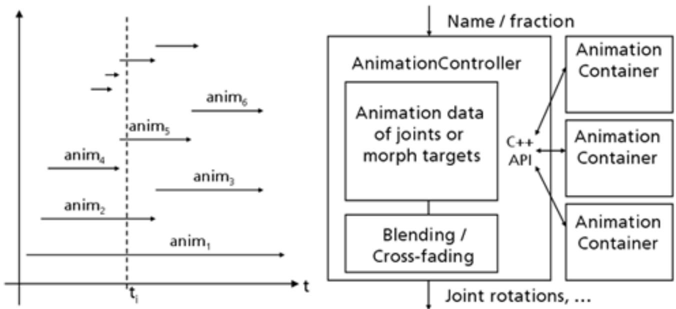

Figure 5: Timeline at time t_i, and mixing in AnimationController.

```
DEF comp TimelineComposer {}
```

As already mentioned in section 7.1, this component is called behavior controller instead of behavior creation. Thus, almost all behavior or animations respectively you want to use, first need to be defined and specified within X3D. If, for example, you want to define a character with animations 'a' and 'b', then you need a file that contains the character's geometry (e.g. a Transform with an HAnimHumanoid child node) as well as files containing the AnimationContainer nodes for 'a' and 'b'. Thus, for almost all actions you want to trigger, there has to exist an AnimationContainer that holds all animation data or at least references to certain nodes and scene data.
As previously mentioned, the latter can have two types: The InstantAnimationContainer is referenced by actions that have no duration (i.e. transitions with dur='0') – e.g. by a `<transform>` in an actions script or a `<fragment>` in the definitions script respectively. The TimedAnimationContainer simply contains the animation's default duration as well as all Interpolator nodes etc. for the desired animation. Please note here, that the order in which the fields 'interpolators', 'fieldnames', and 'targetnames' appear is important: the 1st interpolator is used for the 1st from 'targetnames'.'fieldnames' and so on. The concrete mapping from these animation containers to certain PML commands can then be achieved by using a so-called Gesticon or at least some type of asset management (which has to be set-up in advance) to achieve maximum portability.

References

[1] Stefan Göbel, Oliver Schneider, Ido Iurgel, Axel Feix, Christian Knöpfle, and Alexander Rettig. Virtual human: Storytelling and computer graphics for a virtual human platform. In *TIDSE 2004*, pages 79 – 88, Darmstadt, 2004.

[2] Yvonne Jung and Johannes Behr. Extending H-Anim and X3D for advanced animation control. In Stephen Spencer, editor, *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology*, pages 57–65, New York, USA, 2008. ACM Press.

[3] Yvonne Jung and Johannes Behr. Simplifying the integration of virtual humans into dialog-like VR systems. In *Proc. of IEEE Virtual Reality 2009 Workshop: 2nd Workshop on Software Engineering and Architectures for Realtime Interactive Systems*, pages 41–50. Shaker, 2009.

[4] Yvonne Jung and Christian Knöpfle. Real-time rendering and animation of virtual characters. *The International Journal of Virtual Reality (IJVR)*, 6(4):55–66, 2007.

[5] Martin Klesen and Patrick Gebhard. Affective multimodal control of virtual characters. *IJVR*, 6(4):43–54, 2007.

[6] Christian Knöpfle and Yvonne Jung. The virtual human platform: Simplifying the use of virtual characters. *The International Journal of Virtual Reality (IJVR)*, 5(2):25–30, 2006.