AAF To XTL conversion for Window Media Series 9

Prepared for: BBC Technology

Prepared by: Jim Trainor
James Trainor Engineering Inc.
519-896-7165
j.trainor@rogers.com

Revision History:

- Initial revision, 3 Oct 2003

1 Introduction

XTL is Microsoft's representation of editing metadata and is supported by Windows Media Series 9. XTL represents sequences of audio and video segments with transitions and effects. AAF files represent the same thing. XTL provides the means by which AAF files can be supported by Microsoft Windows Media 9.

To support an AAF file in Microsoft Windows Media, software must be developed to:

- Convert AAF editing metadata to its XTL equivalent.
- Provide a means to access audio and video essence that is embedded inside an AAF file.

This document addresses only the former: AAF to XTL metadata conversion.

Familiarity with both AAF and XTL is assumed.

XTL is a set of XML elements with containment relations represented informally in figure 1. An XTL timeline contains one or more groups (i.e. audio or video), each group contains a composition that is comprised of any number of tracks (i.e. the essence data), transitions, effects, and other compositions.

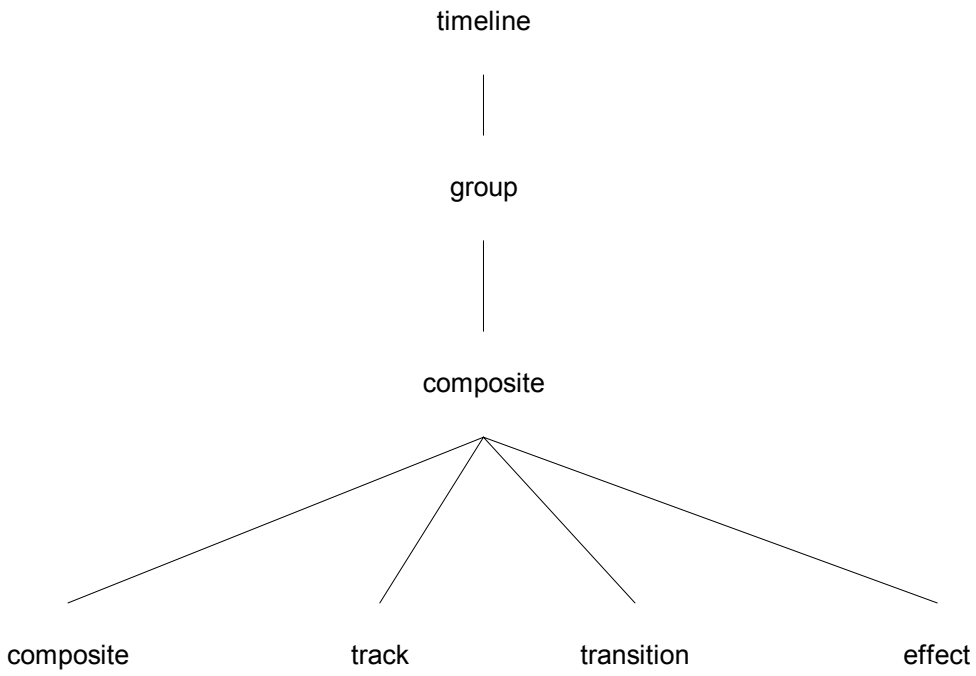

Figure 1 – Simplified representation of XTL element containment.

The AAF object model represents a nearly identical structure, represented informally in Figure 2. An AAF file contains a CompositionMob, the composition has one or more MobSlots (which may represent audio or video), each MobSlot has a Sequence, and a

Sequence is comprised of any number of SourceClips, Transitions, OperationGroups, and other Sequences.

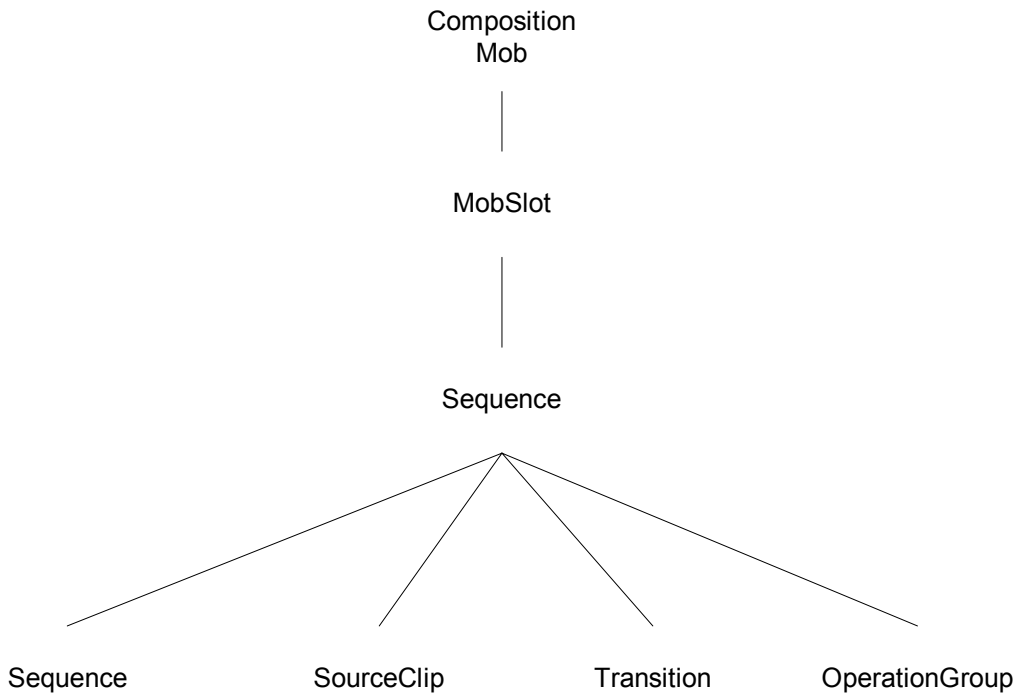

Figure 2 – Simplified representation of AAF object containment.

The parallels between the two are clear when represented this way.

This software implements AAF to XTL conversion by building a tree representing AAF object containment (and other) relationships. Following this, the tree is traversed a number of times to implement processing operations as required to bring it to a state where its structure suitably represents that of XTL, and all required information to generate XTL is available.

2 Design and Implementation

A small framework called "Aaf Input Framework" exists to support this representation of an AAF file and the processing operations required to implement XTL conversion. The "Aaf Input Framework" consists of:

- A base class for nodes in the tree.
- A base class for visitors that implement processing operations on nodes in the tree.
- A base class for objects that can decorate nodes of the tree.
- A tree builder that will generate an AifParseTree (see UML in figure 3) rooted in any object in an AAF file.

The “visitor” and “decoration” concepts are variations of the design patterns described in the well known book: “Design Patterns” by Gamma, Helm, Johnson, and Vlissides. These software design concepts are not described in this document.

The input framework is packaged in a library called “aiflib”. Any class prefixed with “Aif” is part of this library.

Another library, “aif2xtl” implements specialized tree nodes, visitors, and decorations, and provides the means to take a basic tree representing an AAF composition and process it to produce XTL. The class prefix for classes in this library is “Aif2Xtl”.

A simple command line program called “aaf2xtl” uses these libraries to implement a small aaf to xtl utility.

2.1 Tree and Tree Nodes

Figure 3 represents, in UML, the structure of the classes that represent nodes in the tree, and the tree itself.

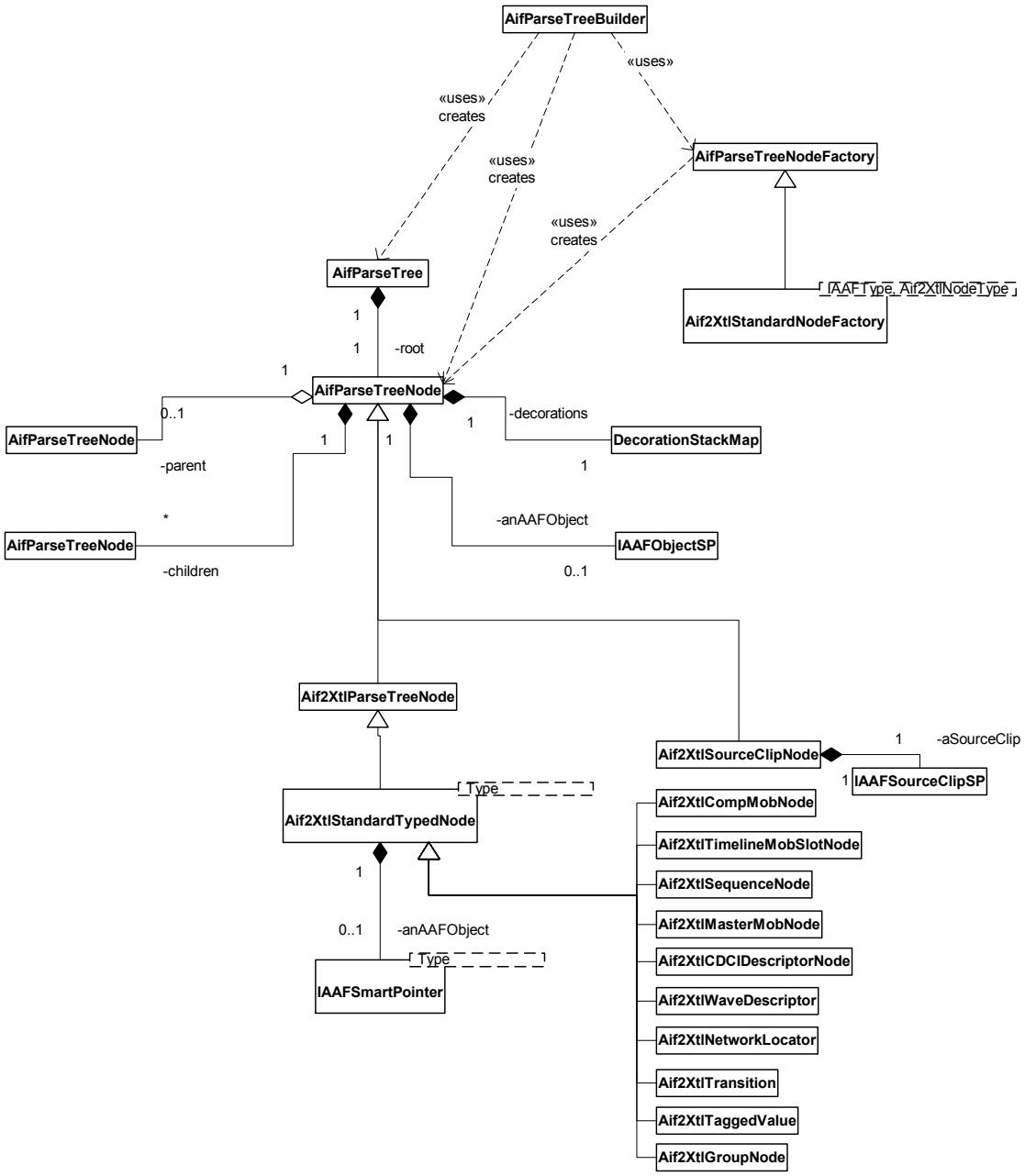

Figure 3 – UML representing the structure of Aif, and Aif2Xtl, tree nodes and related classes.

Node Class Synopsis	
AifParseTreeBuilder	This is a small utility class that uses the "recursive iterator" implemented in AxLib to build a tree of AAF objects. The class in axLib is AxBaseObjRecIter, found in the example2/axLib directory included with AAF SDK. (Note, the current implementation is actually just a function, not a proper class.)
AifParseTreeNodeFactory	This is the factory base type used by AifParseTreeBuilder to create tree nodes for each AAF object it encounters in a file. It has a default implementation that creates instances of generic AifParseTreeNode objects.
Aif2XtlStandardNodeFactory	This is the specialized factory implementation used to create specialized tree nodes to represent the AAF objects we are interested in insofar as XTL conversion processing is concerned.
AifParseTree	This is a small class that contains the root node of a tree. One could easily forgo its use and simply use the root node directly.
AifParseTreeNode	The base class for all nodes in the parse tree. This object optionally stores an AAF object. There is currently only one instance where no AAF object is stored. That is the Aif2XtlGroupNode. A node has a parent (null for the root node) and an ordered set of children.
Aif2XtlParseTreeNode	Specialized version of AifParseTreeNode used to represent AAF objects, and other nodes, that require special treatment insofar as XTL conversion processing is concerned.
Aif2XtlStandardTypedNode	This is a standard templated implementation of Aif2XtlParseTreeNode.
Aif2XtlSourceClipNode	AAF SourceClips require special treatment that could not be accommodated by Aif2XtlStandardTypedNode. This class exists to support that special treatment.

Node Class Synopsis	
DecorationStackMap	This supports decoration objects of arbitrary type and number that can be pushed onto any node instance. This is actually a typedef. The defined type is map<char*,stack<AifParseTreeNodeDecoration *>>. The map key (char*) is the type name of any object derived from AifParseTreeNodeDecoration.

The parse tree is rooted in the first CompositionMob found in the AAF file. A typical tree produced by the builder will look something like that in figure 4.

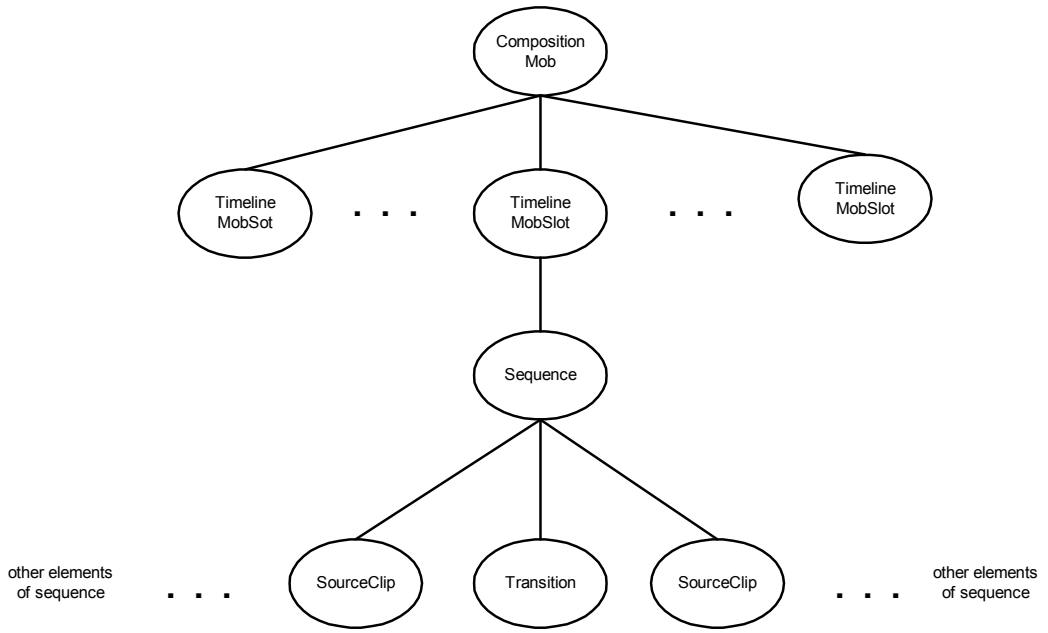

Figure 4 – Typical tree of AAF objects, rooted in a CompositionMob, produced by AifTreeBuilder.

Note that the leaves of the tree in Figure 4 are SourceClip objects. The SourceClips have a reference to a MobSlot in a MasterMob that in turn refers to a chain of SourceMob objects. The builder does not create a tree than includes the MasterMob and SourceMob objects because these objects are only referenced, not contained, by the SourceClip object in the AAF file.

A visitor implementation is used to locate all the SourceClip objects, resolve the MasterMob and SourceMob references, and add these objects to the parse tree.

2.2 Visitors

Figure 5 represents, in UML, the structure of the visitor classes.

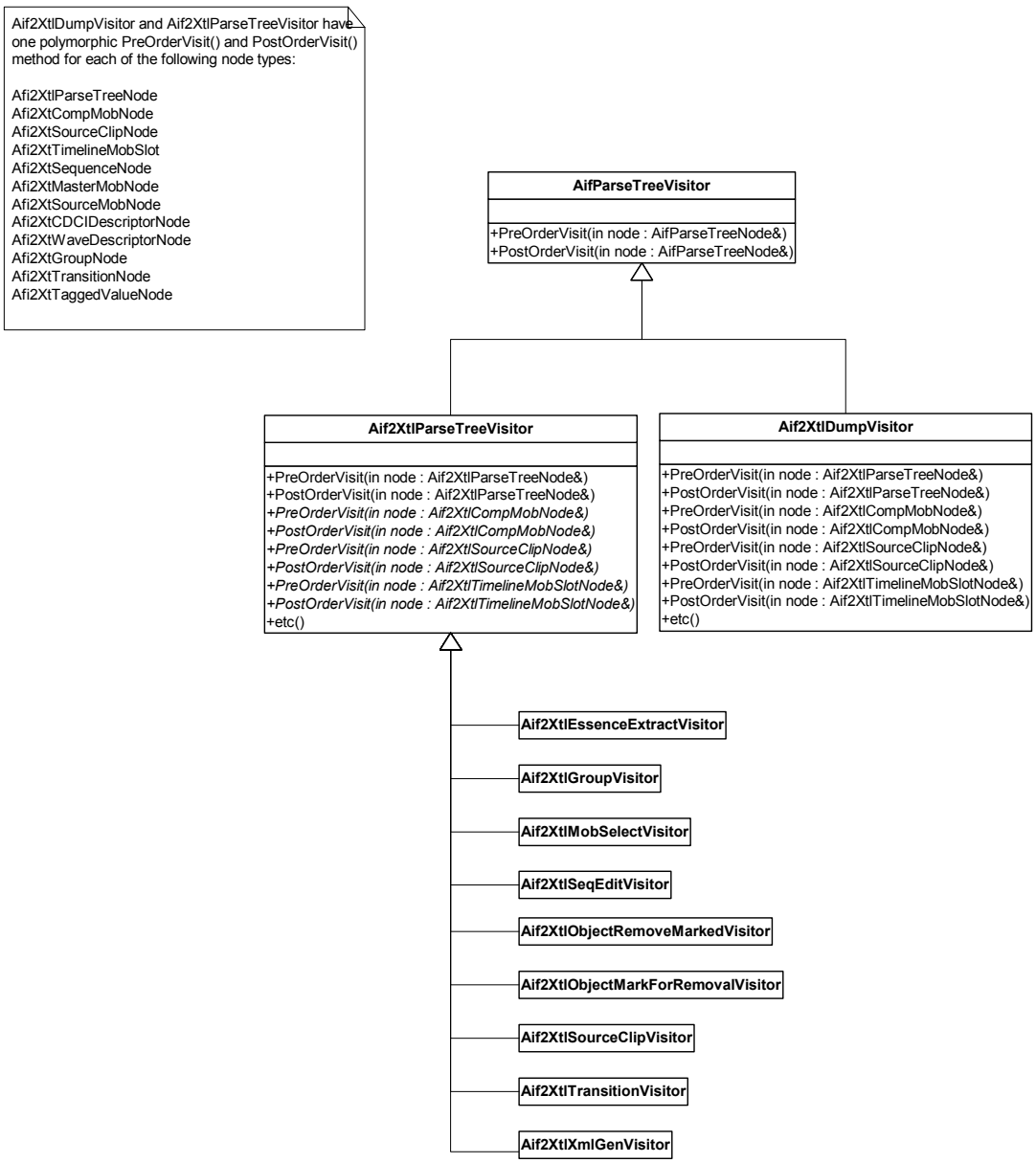

Figure 5 – UML representing the structure of Aif visitor base class and derived Aif2Xtl visitor implementations.

Visitor Class Synopsis	
AifParseTreeVisitor	Visitor base class.
Aif2XtlParseTreeVisitor	Visitor implementation that extends AifParseTreeVisitor by adding polymorphic Visit() methods for each of the node types defined in the Aif2Xtl library.

Visitor Class Synopsis	
Aif2XtlGroupVisitor	Inserts one audio and one video group node as children of the composition mob node and reparents the composition mob's timeline slots accordingly.
Aif2XtlSourceClipVisitor	Visits a composition's SourceClip nodes and resolves the referenced MasterMob. It then follows the chain of SourceMobs referenced by the MasterMob and adds each SourceMob in the chain as a child of the MasterMob's SourceClip.
Aif2XtlMobSelectVisitor	This selects one mob in each mob chain for further processing.
Aif2XtlTransitionVisitor	Visits transitions and determines if the transitions are supported by XTL. The cut point is also read from the AAF object and saved for possible later use.
Aif2XtlSeqEditVisitor	This computes the edit points along the XTL timeline for each segment (e.g. SourceClip) in each sequence in the tree. The visitor processes both cuts and transitions, and determines the segment's start and stop points. These start/stop points are later used by the Aif2XtlXmlGenVisitor.
Aif2XtlEssenceExtractVisitor	This visits all SourceMobs and determines if the referenced essence data is in the AAF file. If it is, the essence is extracted and written to an external file. *This is a work around that is required until a plugin exists for Windows Media that can read the essence directly out of the AAF file.*

Visitor Class Synopsis	
Aif2XtlObjectMarkForRemovalVisitor	Some objects in the AAF file are not required and can cause problems if not removed from the parse tree. In particular, the TaggedValue objects that appear in Avid's AAF files. This visitor marks (i.e. places a decoration on) these objects so that they can be removed by the Aif2XtlObjectRemovedMarkedVisitor
Aif2XtlObjectRemovedMarkedVisitor	This visitor locates all objects that are marked for removal and removes them from the tree.
Aif2XtlXmlGenVisitor	This visitor generates the XTL output. It writes it to an ostream supplied as an argument of the class' constructor. This visitor depends on the results of the processing operations executed by all previous visitors. It will look for node decorations that have the information it requires to generate the xtl, e.g.: editing information, transition information, the location of extracted essence, etc.
Aif2XtlDumpVisitor	Dumps a text representation of the parse tree. Used for debug only.

The Aif2XtlGroupVisitor and Aif2XtlSourceClipVisitor modify the tree as described in the synopsis above. Figure 6 shows the additional nodes that are created, and inserted, by the Aif2XtlGroupVisitor and the Aif2XtlSourceClipVisitor. Aif2XtlSourceClipVisitor uses the AifParseTree builder to create the subtrees that become children of each SourceClip.

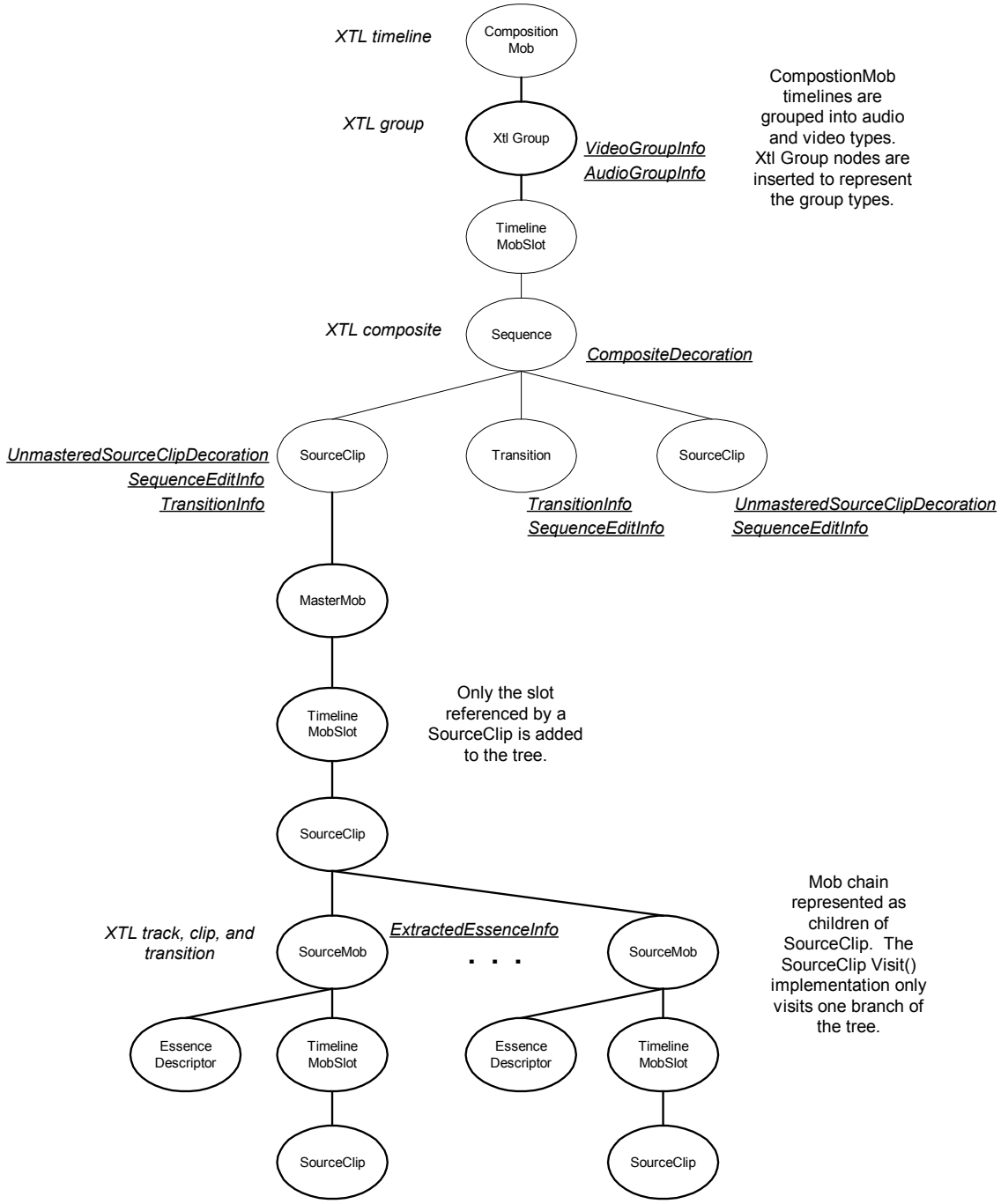

Figure 6 – The tree shows the nodes added by the Aif2XtlGroupVisitor and Aif2XtlSourceClipVisitor. Also shown, underlined, are the decorations placed on the nodes, and used by, various visitor implementations. The nodes that directly correspond to XTL output generated by the Aif2XtlXmlGenVisitor are also noted.

The visitor interface has polymorphic PreOrderVisit() and PostOrderVisit() methods. This means that for each node, the visitor is called before and after the child nodes are visited. This is used, for example, by the Aif2XtlXmlGenVisitor to generate opening and

closing XML syntax. In the case of the CompositionMob node, the Aif2XtlXmlGenVisitor::PreOrderVisit(Aif2XtlCompMobNode&) method is called and outputs <*timeline*>, the child nodes are traversed and generate their XTL, then Aif2XtlXmlGenVisitor::PostOrderVisit(Aif2XtlCompMobNode&) is called and generates the </*timeline*> closing syntax.

2.3 Decorations

Decorations are a simple means to attach arbitrary information to a node in the tree. Any number of decorations of any type can be attached a node. Each node maintains a stack for each type of decoration that is pushed onto the node. Decorations must derive from AifParseTreeNodeDecoration. Visitor implementations create instances of concrete decorations and push those onto nodes. Subsequent visitors can check if a decoration of a particular concrete type exists, and optionally pop it off the node's decoration stack.

To manage multiple decoration types, AifParseTreeNode maintains a map of stacks of AifParseTreeNodeDecoration pointers. The map key is a string that is the type name of the concrete decoration type stored in the stack. The type name is generated using the runtime type info provided by the C++ typeid operator.

The complexity of managing multiple decoration types if hidden in the templated AifParseTreeNode IsDecorate(), GetDecoration(), PushDecoration(), and PopDecoration() methods. Users of the decoration methods, of course, have are not exposed to the type name mapping details.

Figure 6 is the UML representation of the current set of decoration implementations.

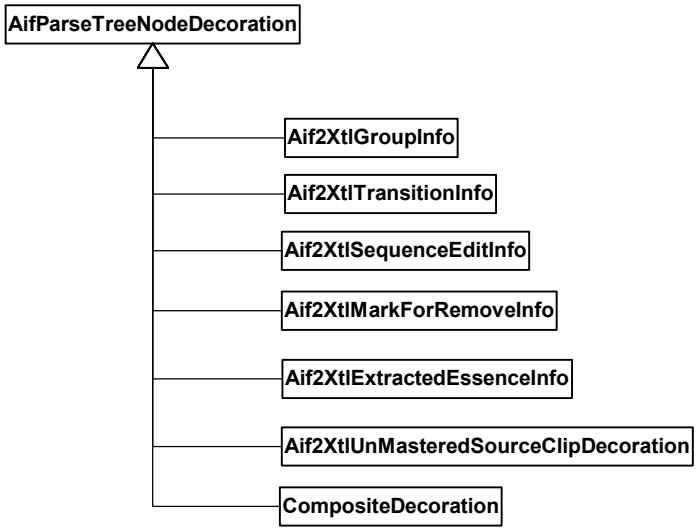

Figure 7 – Node decoration classes.

Decoration Class Synopsis	
AifParseTreeNodeDecoration	Base class for all decorations.

Decoration Class Synopsis	
Aif2XtlGroupInfo	Attached to Aif2XtlGroupNode objects. Stores the group type: audio or video.
Aif2XtlTransitionInfo	A transition info decoration is created and attached to a Aif2XtlTransitionNode by Aif2XtlTransitionVisitor, and moved to the preceding Aif2XtlSourceClipNode by Aif2XtlSequenceEditInfo. It stores information required to generate XTL transition output.
Aif2XtlSequenceEditInfo	This object is placed on each segment in a sequence. It contains a segment's start position, end position, essence data offset, and edit rate, required by the Aif2XtlXmlGenVisitor.
Aif2XtlMarkForRemoveInfo	This is placed on nodes by the Aif2XtlObjectMarkForRemovalVisitor. The Aif2XtlObjectRemoveMarkedVisitor then removes nodes tagged with the decoration.
Aif2XtlExtracedEssenceInfo	This decoration is pushed onto Aif2XtlSourceMobNode nodes and identifies the name of the external file where extracted essence data is stored.
Aif2XtlUnMasteredSourceClipDecoration	At various stages in the processing it is useful to distinguish source clips that are "unmastered" (i.e not an ancestor of a master mob). This decoration is used for that purpose. It is placed on these nodes by Aif2XtlSourceClipVisitor before the master mob sub trees are built and appended to the tree.
CompositeDecoration	This is used by Aif2XtlXmlGenVisitor to implement a processing operation that avoids outputting empty XTL <composite><\compostion> elements for empty sequences, e.g. a sequence that contains only an AAF Filler object.

3 Example

The file demonstrated at the IBC 2003 press conference is:

```
samples/on_site_ibc03/press_conference_from_avid.aaf
```

This is a short sequence of 5 segments without transitiitions (i.e. cuts only). The following command generates XTL describing this sequence:

```
aaf2xtl -file samples/on_site_ibc03/press_conference_from_avid.aaf
```

The resulting xtl output is in Appendix 1. If the output is redirected to a file, the Windows Media 9 player can play that file. Read the instructions in aif/msxtl/ReadMe concerning xtl file support in the player.

This file has embedded DV video essence and embedded wave audio essence. Currently, no means exists for the Windows Media 9 player to read embedded essence. Until such support exists, the essence must first be extracted. This only needs to be done one time. The command is:

```
aaf2xtl -file samples/on_site_ibc03/press_conference_from_avid.aaf -extess
```

By default, this writes the extracted essence files to c:/tmp/aaf2xtl_essence. The file name is constructed using the SourceMob (and EssenceData) ID. The video essence is assumed to always be in DV format and the file is suffixed with ".dv". Windows Media 9 does not support .dv essence in this form. The .dv files must be converted to .avi before the resulting XTL will successfully play using the Windows media player. For the purposes of the demonstration at IBC 2003, this conversion was done using a utility called "DVFileConverter" available at http://www.dvunlimited.com.

If the AAF file is passed as the only argument to aaf2xtl a temporary xtl file is written to c:/tmp/aaf2xtl.xtl and a Windows media player process is created to play this file. The command is simply:

```
aaf2xtl samples/on_site_ibc03/press_conference_from_avid.aaf
```

This will play successfully only if the essence data has already been extracted.

4 Appendix 1 – Sample xtl file generated by aaf2xtl.

This is the xtl representation of the AAF file used for the IBC 2003 AAF interoperability press conference. It was generated with the command:

```
aaf2xtl -file samples/on_site_ibc03/press_conference_from_avid.aaf
```

Note: The filenames contain full mob IDs. They have been shortened to fit on a single line here.

```
<timeline>

    <group type="audio">
      <composite>
            <track>
              <clip
               src="c:/tmp/aaf2xtl_essence/3f60935c-1625-001b.wav"
                 start="0:0:0.000" stop="0:0:7.040"
                  mstart="0:0:0.000"/>
```

```
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f609363-31ac-001b.wav"
           start="0:0:7.040" stop="0:0:12.240"
            mstart="0:0:0.000"/>
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f609361-2b34-001b.wav"
           start="0:0:12.240" stop="0:0:19.679"
            mstart="0:0:0.000"/>
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f60935f-22c8-001b.wav"
           start="0:0:19.679" stop="0:0:24.399"
            mstart="0:0:0.000"/>
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/03f60935d-1c11-001b.wav"
           start="0:0:24.399" stop="0:0:28.920"
            mstart="0:0:0.000"/>
      </track>
</composite>
<composite>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f60935c-1683-001b.wav"
           start="0:0:0.000" stop="0:0:7.040"
            mstart="0:0:0.000"/>
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f609363-3219-001b.wav"
           start="0:0:7.040" stop="0:0:12.240"
            mstart="0:0:0.000"/>
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f609361-2b92-001b.wav"
           start="0:0:12.240" stop="0:0:19.679"
            mstart="0:0:0.000"/>
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f60935f-2316-001b.wav"
           start="0:0:19.679" stop="0:0:24.399"
            mstart="0:0:0.000"/>
      </track>
      <track>
        <clip
          src="c:/tmp/aaf2xtl_essence/3f60935e-1c6f-001b.wav"
           start="0:0:24.399" stop="0:0:28.920"
            mstart="0:0:0.000"/>
      </track>
```

```
      </composite>
    </group>
  <group type="video" width="720" height="576" framerate="25.0000">
    <composite>
          <track>
            <clip
              src="c:/tmp/aaf2xtl_essence/3f60935a-0fcd-001b.avi"
               start="0:0:0.000" stop="0:0:7.040"
                  mstart="0:0:0.000"/>
          </track>
          <track>
            <clip
              src="c:/tmp/aaf2xtl_essence/3f609362-2c5d-001b.avi"
               start="0:0:7.040" stop="0:0:12.240"
                  mstart="0:0:0.000"/>
          </track>
          <track>
            <clip
              src="c:/tmp/aaf2xtl_essence/3f60935f-23c2-001b.avi"
               start="0:0:12.240" stop="0:0:19.679"
                  mstart="0:0:0.000"/>
          </track>
          <track>
            <clip
              src="c:/tmp/aaf2xtl_essence/3f60935e-1d1a-001b.avi"
               start="0:0:19.679" stop="0:0:24.399"
                  mstart="0:0:0.000"/>
          </track>
          <track>
            <clip
              src="c:/tmp/aaf2xtl_essence/3f60935c-172f-001b.avi"
               start="0:0:24.399" stop="0:0:28.920"
                  mstart="0:0:0.000"/>
          </track>
    </composite>
  </group>
</timeline>
```