An Efficient, Streamable Text Format for Multimedia Captions and Subtitles

Dick C.A. Bulterman, Jack Jansen, Pablo Cesar, Samuel Cruz-Lara

An Efficient, Streamable Text Format for Multimedia Captions and Subtitles

Dick C.A. Bulterman, A.J. Jansen, Pablo Cesar
CWI: Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands
{Dick.Bulterman, Jack.Jansen, P.S.Cesar}@cwi.nl

and

Samuel Cruz-Lara
INRIA Lorraine
Laboratoire Loria BP 239
F-54506 Vandoeuvre Les Nancy cedex
France
Samuel.Cruz-Lara@loria.fr

ABSTRACT

In spite of the high profile of media types such as video, audio and images, many multimedia presentations rely extensively on text content. Text can be used for incidental labels, or as subtitles or captions that accompany other media objects. In a multimedia document, text content is not only constrained by the need to support presentation styles and layout, it is also constrained by the temporal context of the presentation. This involves intra-text and extra-text timing synchronization with other media objects. This paper describes a new timed-text representation language that is intended to be embedded in a non-text host language. Our format, which we call *aText* (for the Ambulant Text Format), balances the need for text styling with the requirement for an efficient representation that can be easily parsed and scheduled at runtime. *aText*, which can also be streamed, is defined as an embeddable text format for use within declarative XML languages. The paper presents a discussion of the requirements for the format, a description of the format and a comparison with other existing and emerging text formats. We also provide examples for *aText* when embedded within the SMIL and MLIF languages and discuss our implementation experiences of *aText* with the Ambulant Player.

Categories and Subject Descriptors

D.3.2 [**Language Classifications**]: Specialized application languages; I.7.2 [**Document and Text Processing**]: Document Preparation—Languages and systems.

General Terms

Performance, Design, Experimentation, Standardization, Languages.

Keywords

Timed text, SMIL, DFXP, RealText, streaming text, Ambulant.

1. INTRODUCTION

XML multimedia languages integrate a collection of audio, graphics, image, text, and video media items into a single presentation. Since most media items are by nature very large — they consist of sampled data that can run into the gigabytes — it has been common wisdom in multimedia document design to develop container formats that refer to media objects by reference. A good example of such a language is SMIL [4], which does not contain any actual media data but consists of scheduling and layout directives that control the presentation of a set of external (to the document) media object URIs.

While the clean separation of content from document structure is a valid abstract concept, it often brings with it an increased authoring burden. Instead of maintaining (or generating) a single document, several documents need to be defined: one for the integrating structure and one each for the media objects. It also brings media access problems — especially in mobile environments — where multiple transfer sessions need to be initiated to fetch a few bytes of content. While this is defendable for large, typically binary media objects, it is less appropriate for text data.

One approach for increasing authoring and processing efficiency is to integrate an existing text format into the XML container language. While many public and proprietary text formats exist, they cannot be simply imported into a multimedia container format because most are extremely feature rich (meaning that they need a complex text processing engine to manage text styling and layout) and because they were designed to be timing agnostic (meaning that there is no existing syntax for describing intra-text timing, nor extra-text synchronization with other objects). Even dedicated languages for describing timed text (such as RealText [14], DFXP [1], HTML+TIME [10] or MPEG4-Part 17 [7]) often are based on temporal models that clash with the host timing model used in the media presentation.

This paper presents a new language for integrating text in an (processing) efficient and streamable manner into XML-based multimedia languages. We call this language the Ambulant Text Format, or *aText*. Rather than being designed to be a stand-alone text container format, *aText* was designed to be an embedded timed text description format that could extend existing XML languages such as SMIL and MLIF [5].

The principal contribution of *aText* is that a balance has been achieved between a text format that meets the stylistic needs of a set of common multimedia applications classes and one that is easy to parse, schedule and render on a wide range of multimedia implementation platforms (including low-powered mobile devices and low-end set-top boxes). The format has been implemented and integrated into the Ambulant open media player [3].

This paper is structured as follows. Section 2 provides a summary of the principal uses of a temporal text format in multimedia documents. Section 3 surveys existing support for timed text, both within document containers and as stand-alone timed text languages. Section 4 describes the *aText* format in detail, including its design goals, temporal characteristics and the styling facilities available in the language. We also discuss the use of *aText* as an external container format. Section 5 provides an overview of our current implementation status and the performance issues encountered when implementing *aText* within the Ambulant player. We conclude with our expectations for further development and standardization of *aText*.

2. APPLICATIONS CLASSES FOR TEXT IN MULTIMEDIA DOCUMENTS

The functionality proposed for *aText* is based on a set of general use cases that we have found to be common when authoring multimedia presentations.

We preface this discussion with the observation that, unlike languages such as XHTML [12], text often plays an incidental role in multimedia presentations. In text-centric presentations, the flow of text in a document will dominate layout and styling concerns. In multimedia documents, the temporal nature of continuous media object (and the presentation sequence of objects such as graphics elements and images) define the presentation structure. When text is present, it normally plays a subordinate role.

The subordinate role of text does not mean that it is unimportant. The appropriate application of text can introduce clarity that might otherwise be missing. It is often said that a "picture is worth a thousand words"; what is often forgotten is that without some form of descriptive (text) context, a picture alone may be worthless.

We have identified the following eight classes of incidental text content as being common in multimedia applications:

1. *Headlines*: this class requires support for relatively short strings of text that can be stylistically differentiated from other text. This differentiation may be based on size, color or placement (centered vs. side-aligned).
2. *Labels*: this class requires support for relatively short strings of text. The text may need to be of varying sizes for various uses, such as photo captions or button labels. Significant text formatting is typically not required.
3. *Captions/subtitles*: this class requires support for multi-line strings of text that provide a visual representation of both spoken text and audio cues in content that is being presented in parallel with the text. It may also require style cues for speaker identification. This class will require light formatting (alignment, line breaks) and tight temporal coupling with other objects in the presentation. Historically, captions and subtitles have been geared for use by the hearing impaired.
4. *Foreign-language subtitles:* this class is similar to the captions/subtitles described in class #3, except that audio cues and speaker identification are typically not presented; there is an assumption that the foreign-language reader can see and hear the speakers and cues.
5. *Time-constrained moving text*: this class requires support for marquee-style crawling or credits-style rolling text. The content may or may not be directly related to other content in the presentation (such as an independent news crawl under an unrelated video). Such moving text needs to have a rate associated with it that is not related to its internal content. It may also need to loop based on external (to the text) factors.
6. *Inter-object triggered text*: this class requires the specification of text fragments that are either triggered by external media (such as a particular frame in a video) or that trigger external objects (such as the sound of a gong when a certain word appears in the text). The level of granularity may be at the word or phrase level, rather than at the text object level.
7. *Conditional timed text*: this class requires support for the conditional specification of text into a rendering area. The conditions may depend on the state of variables in the integrating document, and may be dynamically evaluated during the presentation of the text.
8. *Static block text*: this class requires support for relatively large amounts of formatted text. The text block may consist of informative information that is conditionally presented during a presentation, such as help text or content descriptions.

As we will see, *aText* was designed to provide substantial support for the first seven classes and reasonable support for the eight class of text.

3. EXISTING TEMPORAL TEXT FORMATS

Before deciding to develop *aText* as a new format, we surveyed principal timed-text mechanisms to see if they could be adapted for the applications classes described above. These existing formats fall into two broad categories:

- *External text formats*: formats in which all text styling and timing control is contained in an external media object.
- *Embedded text formats:* formats in which text styling and timing control is interspersed within the parent container format.

This section reviews the structure and characteristics of representative embedded and external timed text formats.

3.1 External Text Formats

External timed text formats provide a self-contained text content, styling, positioning and timing model. They are referenced in a presentation as if they were a video or audio object. External formats represent the most prevalent forms of timed-text support.

When using an external text container, the host document will often specify a rendering area into which the content is displayed and an explicit duration for the text object. The specified duration may be shorter or longer than a duration specified in the external file. In these cases, the timing within the host document will determine the amount of text displayed. Similarly, the rendering area may be larger or smaller than that required by the external text object.

3.1.1 W3C DFXP

DFXP, the *distribution format exchange profile* of W3C Timed Text [1], is an emerging standard for encoding richly styled text content. DFXP was designed to meet a broad range of text needs while still being simple to hand-author.

Timing control in DFXP is based on SMIL 2.1. There is support for SMIL's sequential <seq> and parallel <par> timing, but not for the exclusive time container <excl>. Timing is specified using attributes on the <body>, <div>, <p>, <region> and <span> elements.

Text can be styled using a broad range of XSL 1.0 [2] styling parameters. Text can be dynamically reflowed within a rendering extent and several of the style properties can be animated with the SMIL Animation <set> element.

The following fragment gives an example of DFXP:

```
<tt xml:lang="" xmlns="http://www.w3.org/2006/10/ttaf1">
 <head>
   <metadata/>
   <styling/>
   <layout/>
 </head>
 <body region="subtitleArea">
   <div>
     <p xml:id="subtitle1" begin="0.76s" end="3.45s">
       I think that I shall never see ...
     </p>
     <p xml:id="subtitle2" begin="5.0s" end="10.0s">
       ... a word that rhymes with<br/>
       <span fontColor="orange">orange</span>!
     </p>
   </div>
 </body>
</tt>
```

For most applications of plain text, DFXP is not backward compatible with XHTML. It uses a layout model and an attribute structure that is not compatible with SMIL.

3.1.2 RealNetworks RealText

RealText [14] is a popular proprietary streaming text format that was designed to meet an intermediate range of text formatting applications. It can be automatically generated or hand-authored.

Timing control in RealText is based on a timing model that predated SMIL. Text is structured as a set of temporal fragments that can be scheduled based on explicit time codes placed in the document. RealText predefines a set of common text rendering models, such as *marquee*, *teleprompter*, *tickertape*. Simple content structuring and styling is provided within a temporal block.

An example of RealText encoding of timed text is shown in the following fragment:

```
<window type="generic" duration="1:0.0" scrollrate="0"
      width="500" height="390"bgcolor="#000080">
<time begin="0:0.76"/>
I think that I will never see ...
<time begin="0:3.45"/>
<clear/>
<time begin="0:5.0" end="0:10.0"/>
... a word that rhymes with
<br>
<font color="orange">orange</font>!
```

In RealText, the end attribute defines a document-global end time. Once specified, it applies to all subsequent objects unless redefined. This is a unique (if not particularly useful) feature.

3.1.3 SRT

Most of the text formats discussed in this article are relatively formal, in the sense that they have been developed, published and maintained by academic, commercial or public organizations. An exception is a popular but un-sanctioned foreign-language subtitle format typically known as SRT. SRT has its origins in a grassroots effort to provide subtitles for commercial DVD content in languages not readily supported by major production studios. A brief history of SRT is available on Wikipedia [15]. We include SRT because of it practical impact: hundreds of thousands of .srt encoded subtitles files are available to internet users.

SRT provides a simple model for scheduling captions. Each caption entry contains a caption number, a display and removal time, one or more text lines (each line forces a line break) and a blank line delimiter. The follow example provides a complete summary of SRT capabilities:

```
1
 00:00:00,760 --> 00:00:03,450
I think that I shall never see ...

 2
 00:00:05,000 --> 00:00:10,000
A word that rhymes with
orange!
```

SRT does not provide support for text styling or layout; these are provided by the media player.

3.1.4 MPEG4-17/3GPP Streaming Text

Where SRT represents an underground timed text format, MPEG-4 Part 17 Streaming Text [7] (which we call MP4TT for short) is an example of a high-brow, studio-quality format. MP4TT was designed to provide exact placement of captions and foreign-language subtitle text on top of commercial production content. Where SRT (and, to lesser extents, DFXP and RealText) provide best-effort positioning and styling, MP4TT leaves little room for individual rendering agents.[1] Text quality is elevated to the level of commercial audio and video. That being said, MMP4TT is aimed almost exclusively at simple support for captions/subtitles that are to be superimposed on other MPEG-4 objects. A good example is DVD captions. This simple format has also been adopted by the 3GPP for use in mobile telephones [16].

MP4TT (and 3GPP mobile text) consists of a sealed container model in which text properties are not hints but fixed directives. There is limited ability for styling and for positioning of text. Timing is specified using timecodes placed into the stream representation.

MP4TT is a binary format. There is no text representation for MP4TT; as a result, an example is not provided here.

3.2 Embedded Text Formats

Embedded timed text formats provide a text content, styling, positioning and timing model that is integrated into the host language. At present, there is no widely available embedded text format that addresses all of the applications class discussed in section 2. The two prevalent open formats that provide some embeddable features, SMIL and Microsoft's HTML+TIME, are discussed in this section.

3.2.1 Timed Text in SMIL

All versions of SMIL have had support for the <text> element, which is defined as an alias of the generic SMIL <ref> media reference element. SMIL <text> elements have all of the temporal and positioning control available to any SMIL object, but this control is only available on the element container and not on individual text fragments with the object.

A typical use of the text element is:

```
<text region="Title" src="Headline.html" begin="3s" dur="3s" />
```

This construct causes an external HTML renderer to be activated, with the content being rendered in the named region. One problem with this approach is that activating an HTML renderer is typically a heavy-weight operation that on low-resource platforms may take longer than the time allotted for the actual text rendering.

Users new to SMIL are often surprised that the text element does not have a content model—that is, an ability to specify the content text along with the element, such as in:

```
<text region="Title" dur="10s" >
    A Poem About Colors
</text>
```

More advanced users of SMIL found that they were able to insert in-line text content into the SMIL file using a data URL, such as:

```
<text src="data:,A%20Poem%20About%20Colors"
    region="Title" dur="10s"/>
```

1.The exact placement of text blocks ensures that important content — such as paid product placement ads — are not obscured.

The strict syntax of this approach, plus the limited styling options available, make it a less-than-optimal way of including incidental text content into a SMIL presentation. The main benefit of this approach is processing speed.

SMIL provides full timing and layout facilities that assist in the scheduling and placement of text. It also provides a mechanism for passing styling parameters to a text object. A SMIL version of our timed text example is:

```
<smil>
 ...
 <par>
   <text begin="0.76s" end="3.45s" src="L1.txt" region="C"/>
   <text begin="5s" end="10s" src="L2.txt" region="C">
     <param name="fontColor" value="orange">
   </text>
 </par>
 ...
</smil>
```

Note that in this example, the begin time of the element referencing object L2 is relative to the start of the <par>; if a <seq> container had been used, the start time would have been relative to the end of the proceeding object. Note also that the entire text string content for object L2 is colored orange: SMIL cannot send substring styling information.

3.2.2 Timed Text in HTML+TIME

In a SMIL presentation, a single document-wide temporal structure guides the scheduling of document components. In HTML+TIME [10], which is Microsoft's implementation of the W3C XHTML+SMIL candidate recommendation [11], timing support is provided on an incidental basis within an a-temporal document.

HTML+TIME uses a full HTML engine. This provides extensive support for text layout and styling based on CSS [9]. Temporal control is provided using a time container model similar to the approach used in DFXP. The following fragment illustrates the use of HTML+TIME for our poem:

```
<html xmlns:t ="urn:schemas-microsoft-com:time" >
 <head>
   <style>
      .time { behavior: url(#default#time2) }
   </style>
   <?IMPORT namespace="t"
     implementation="#default#time2">
   <title>A Short Poem</title>
 </head>
 <body>
   <div t:timecontainer="par">
     <p class="time" begin="0.76s" dur="3.45s">
       I think that I shall never see ...
     </p>
     <p class="time" begin="5s" end="10s">
       A word that rhymes with<br>
       <span color="orange">orange</span>!
     </p>
   </div>
 </body>
</html>
```

3.3 Comparison of Timed Text Formats

Table 1 summarizes the degree to which the external and embedded timed text formats discussed in this section meet the needs of the applications classes developed in section 2.2. We also provide a column for the results for *aText*.

DFXP and HTML+TIME (and, to a slightly lesser extent, RealText) provide rich facilities for supporting the stylistic needs for headlines and labels, and for supporting blocks of timed text. The main difference between DFXP and HTML+TIME is that DFXP uses a consistent XML styling paradigm with incidental timing, while HTML+TIME focuses on HTML/CSS support and full SMIL timing. RealText provides a non-SMIL temporal model for authoring timed text and more limited styling facilities.

For captions/subtitles and foreign-language subtitles, operations requiring close synchronization between a multimedia object (such as a video or audio object) and text content is only directly supported by MP4TT: it uses a content container model in which both streams are synchronized using a common clock. Other external formats provide adequate support for basic captions timing, but they cannot express direct synchronization relationships with external media objects. Embedded formats like SMIL have the inverse problem: they provide a common time base, but they are not able to provide detailed synchronization of individual text strings with events in a media object in a convenient manner. From a pan-document perspective, HTML+TIME provides a comprehensive range of timing primitives, but it does not provide a container format for multimedia content. (It also requires a very heavy-weight text engine.) SRT is a basic foreign-language captioning format, but it fails to provide the styling support necessary to allow fine-grain text positioning or effective (style-based) speaker or audio cues.

The most extensive support available to date for animated text (animated in the sense of text motion) is RealText. The main restriction imposed by RealText is a fixed set of motion and looping primitives. Neither SMIL, HTML+TIME, DFXP or SRT duplicate this functionality. MP4TT has limited support for fixed types of text motion.

Inter-object activation requires that the timing of individual text fragments can be triggered (or serve as triggers) for items out of the direct scope of the text object. As expected, none of the formats provided extensive support for this feature. External formats cannot influence objects in the host language, and embedded formats do not provide the level of granularity to extensively support this feature. Only HTML+TIME and SMIL provide a limited ability for external, out-of-document events (such as mouse-overs) to trigger internal events, but this support can not generally be applied to or from multimedia objects.

None of the external timed text formats provide direct control for conditional activation of timed text. SMIL provides extensive support for system and custom test variables, all of which can be applied to the activation of text objects. HTML+TIME provides basic support for system text variables, but only via parse-time (and not run-time) evaluation.

The weakest format for supporting the applications classes described in section 2 is SMIL. While SMIL provides extensive timing and conditional activation support, it cannot provide the granularity of control required for supporting captioning/subtitles and foreign language subtitling. While the text required for headlines and labels can be imported from external files, there is little detailed styling control at the character or word level. (This is the reason that images are often used to hold labels and headlines.) The authoring model for text in SMIL is also limited. Addressing these deficiencies was a major goal of our *aText* work.

Our rating of *aText* is shown in the final column of Table 1. Details supporting this rating are given in the next sections.

Table 1: Supporting Application Classes for Timed Text

Use Case	External Timed Text				Embedded Timed Text		*aText*
	DFXP	RealText	SRT	MP4TT	SMIL	HTML+TIME	
Headlines	++	+	-	-	-	++	**+**
Labels	++	+	-	-	-	++	**+**
Captions/Subtitles	+	+	+/-	++	-	-	**+**
Foreign Language Subtitles	+	+	+	++	-	-	**+**
Moving Text	-	++	-	+	-	-	**+**
Inter-Object Activation	-	-	-	-	+/-	+/-	**+**
Conditional Timed Text	-	-	-	-	++	+	**++**
Static Block Text	++	+	-	-	-	++	**+**

4. *aText*: THE AMBULANT TEXT FORMAT

The Ambulant Text (*aText*) facility allows text content to be defined and stored directly within an XML document context of a multimedia host language. There may be many reasons for placing text content in the multimedia source file: for authoring convenience, for distribution simplicity, or for rendering performance.

Perhaps the greatest challenge in defining a (new) text content format is not to identify a useful set of styling and control primitives, but to determine which of these useful primitives can be successfully left out of the format specification. The reasons for being very restrictive on the scope of text content support are many, including:

- Many text content formats already exist; all of these are candidates for use as external text content containers, which means that most of their more interesting features don't need to be duplicated in an embedded format.
- Embedded text should be defined in such a way that it can be efficiently implemented in a wide range of environments, including mobile, desktop and TV set-top boxes. The implementation should not have to rely on an extensive platform-specific text processing facility.
- The initial model should be expandable to more complex text features without having to redefine the base model.

This section describes the *aText* format details. We begin with a consideration of the overall design goals and expected use of our work and then describe the timing and style control features of *aText*.

4.1 Design Goals (and Non-Goals)

The general design goal of *aText* was to define a light-weight text format that could be embedded into one or more XML-based host languages. We have focused heavily on embedding *aText* in SMIL, but our work is not SMIL-specific. The language has also been applied to the emerging ISO standard for multi-lingual text exchange, MLIF [5].

Our intended user community was authors who wanted to add text for each of the applications classes discussed in section 2 to a presentation, or for automatic authoring systems that wanted to automatically generate presentations based on the custom needs of specially targeted users.

The following five features describe the basic temporal functionality of *aText*:

1. *aText* supports the atomic display of a block of text based on the timing constraints defined on the <aText> element by the host language. For SMIL, this means that an <aText> element will be given a begin time and a duration or end time that is defined by the SMIL environment. Within this time interval, the content within the <aText> element should be rendered.
2. Within the interval defined in (1), text should either appear as a whole, or as a sequence of text fragments that are appended to the rendering area based on internal timing directives.
3. During the display of text fragments, it should be possible to explicitly clear the rendering area.
4. During the entire interval defined by (1), it should be possible to have the text content crawl or roll across the display area.
5. The timing attributes for individual text fragments should allow for the specification of absolute, relative and (external) event-based timing constraints.

We also formulated a set of non-goals for the initial specification of *aText*. These included:

1. the removal of selected fragments of text within a block while retaining others;
2. the automatic reflow of text within a block based on removal of embedded fragments; and
3. support for out-of-order temporal specification of text fragments within a block.

The result of these goal and non-goals is the definition of a text format that can be integrated into multimedia languages such as SMIL in an efficient and streamable manner.

4.2 Elements and Attributes

The elements and attributes defined for *aText* are summarized in Tables 2 and 3. The various features are classified as timing, rendering control and styling elements and attributes. We discuss the nature and use of each of the elements and attributes in the following sections.

4.2.1 *Timing Elements and Attributes*

The main temporal element defined by *aText* is the *temporal event value*, or <tev>. The <tev> element allows a timing marker to be inserted into the child text content of the <aText> element. The <tev> element is not similar to a <p>, <div> or <span> element in XHTML or DFXP: it does not encapsulate a range of content, but it defines a temporal moment.

Table 2: Elements Defined for *aText*

Timing	Rendering	Styling
<tev>	<br>	<span>
<clear>		<textStyling>
		<textStyle>

Table 3: Attributes Defined for *aText*

Timing	Rendering	Styling	
begin	textMode	textAlign	textFontFamily
next	textPlace	textWrapOption	textFontSize
id	textMotion	textDirection	textFontStyle
	textMotionRate	textColor	textFontWeight
		textBackgroundColor	

A secondary temporal event element is the rendering space clear element, <clear>. The <clear> element also defines a temporal moment, but one with a special side effect: the entire rendering area associated with the <aText> element is erased. The <clear> has an implied <tev> behavior, in that it defines a marker that temporally delineates any text following the <clear> element.

The <tev> / <clear> elements accept the following attributes:

- begin: this attribute specifies the temporal moment of the marker. The attribute contains a SMIL time value list of one or more non-negative absolute times (relative to the start of the <aText> element) or event-based times.
- next: this attribute is similar to begin, except that any non-event time values are interpretted as non-negative times that are relative to the last-defined <tev> / <clear> element.
- id: the ID attribute allows an identifier to be associated with a particular <tev> or <clear> marker. This id (and the effective timing of the <tev>/<clear>) can be used as an event trigger within other parts of the document.

An example of the use of the begin attribute is:

```
    begin=”12.5s; gong.beginEvent; buzzer.endEvent+0.5s”
```

In this list, the value 12.5s represents an absolute offset from the start of the <aText> element; the gong.beginEvent and buzzer.endEvent+0.5s values represent event triggers. The first time value that resolves to true will result in the activation of the time marker.

It is expected that a common authoring situation will be to define a <tev> / <clear> element with a begin time that is relative to its predecessor (rather than to the <aText> element. For example:

```
    <tev id=”d” begin=”25s”/>
        Please select your answer within 5 seconds.
    <clear begin=”d.beginEvent+5s” />
        Time’s up!
```

As an authoring convenience, this same behavior can be encoded as:

```
    <tev id=”d” begin=”25s”/>
        Please select your answer within 5 seconds.
    <clear next=”5s” />
        Time’s up!
```

When used with the <clear> element, the time value specifies the time at which the clear operation takes place. A <clear> or <tev> element without a begin or next attribute is ignored. If both begin and next attributes are given on the same element, the first one to resolve to true is used. If a time value is specified that resolves to a time in the past, it immediately resolves to true.

Neither the <tev> nor the <clear> elements accept styling attributes. Depending on the host language, either may accept attributes that will allow conditional activation of the time marker. This is discussed in detail in [8].

The following example illustrates the three uses of the use of timing attributes within a <tev> element:

```
    <aText ...>
        This is not
      <tev begin=”1.2s”/><!-- absolute time -->
        the famous roller coaster at Coney Island
      <tev next=”3s”/> <!-- relative time -->
        but the Astoria curve on the
      <tev begin=”gong.beginEvent+0.2s”/><!-- event time -->
        number 7 Flushing line!
    </aText>
```

Since *aText* supports a linear, streamed activation model, the pending events only become in scope once the predecessor <tev> / <clear> has become active. The text rendering will continue when the event has occurred, of when the next <tev> or <clear> with an *absolute* time offset becomes scheduled.

In the examples shown up to this point, the scheduling of time markers depends on the contents of the *aText* element or on external events. It is also possible for a marker to activate external behavior outside of the scope of the *aText* element because each of the <tev>/<clear> containers can contain an externally-visible ID. (This is the reason that we classify the id as a timing attribute.)

The following fragment illustrates the use of a <tev> as an event trigger within a SMIL document:

```
    <smil>
    . ...
      <par>
        <aText ... >
          ...
          <tev id=”doorOpens” next=”22s” />
            Slowly, the door opened...
          ...
        </aText>
        <audio src=”CreakingDoor.mp3”
          begin=”doorOpens.beginEvent” />
      </par>
      ...
    </smil>
```

A side-effect of this structure is that the <aText> element can be used to define an event stream for controlling other parts of a presentation, even if no text content is present. This is exceptionally useful behavior, but it is not the core of the contribution discussed in this article.

4.2.2 Rendering Control Elements and Attributes

The content within the *aText* element is displayed as a set of text fragments, delimited by temporal moments. In order to provide support for a number of common visual effects, *aText* supports one element to explicitly break line content and four attributes to control how the contents of the entire *aText* element are processed.

As with most other XML text languages, the break element <br/> ends the formatting of the present line and continues rendering on a new line:

```
    <aText ...>
      <tev id=”q” next=”0s”/>
        As the boat pulls out,
      <br/>
      <tev id=”r” next=”1.5s”/>
    the arms of the
  <br/>
    passenger terminal
  <tev id="s" next="+2s"/>
  <br/>
    wave goodbye.
</aText>
```

Element "q" illustrates that a <br/> placed after a text fragment will cause a line break at the time that the other text in that fragment is rendered. Element "r" illustrates a line break in the middle of a text fragment. Element "s" illustrates a line break just before a fragment is rendered. Line breaks may be placed at any point in the <aText> content.

Four rendering control attributes are defined for use on the <aText> element:

- textMode: this attribute describes how new fragments are added to content within the same <aText> element. The permitted values are:

  { append | replace | inherit }

  In append mode, each new fragment is appended to the existing text in the rendering area. In replace mode, each new fragment will clear the rendering area before new text is added. This has the effect of replacing every

  <tev begin="x"/>

  with

  <clear next="x"/>
- textPlace: this attribute defines where content is first added to a rendering area. The permitted values are:

  {fromTop | fromBottom | inherit }

  The value fromTop adds fragments starting at the top of the rendering area, with text flowing downward by line. The value fromBottom will place the initial text at the bottom of the rendering area, and then move text content upward one line at a time when each content line is filled (or at an explicit <br/>). Unless the value of the textMotion attribute is roll, this movement will exhibit a jumping behavior. For both fromTop and fromBottom, the host language will define the behavior of rendering when the rendering area is full.[2]
- textMotion: this attribute defines the motion of content within the rendering area. The permitted values are:

  { none | crawl | roll | inherit }

  The crawl attribute causes text to move in as a single line the logical writing direction defined by the textDirection attribute, below. The starting position of the text will be determined by the textAlign attribute, also defined below. The roll attribute causes text to move upward within the rendering region. The initial position of the text is defined by the textPlace attribute.
- textMotionRate: this attribute describes the rate at which text motion occurs. The permitted values are:

  { auto | pixelRateValue | percentageValue | inherit }

  A pixelRateValue define the numer of pixels per second of text movement. A percentageValue defines the roll or crawl rate as a percentage of the duration of the *aText* element (where 100% means that all of the text content will be displayed within the active duration of the *aText* object). The default value is 100%.

2.If SMIL is the host language, the fit attribute will define behavior when the content extends beyond the region. Other XML languages may use the XSL overflow property.

Various control attributes can be combined to achieve a wide range of text rendering effects. For authoring convenience, various combinations may be predefined by a particular profile or by using the <textStyling>/<textStyle> elements, discussed below. For example, by combining rendering attributes, the *window types* defined by RealText [14] can be duplicated. In all cases, the various rendering attributes will be considered as hints to the implementation platform. If a particular platform cannot support the desired type for performance or other reasons, the attributes may be ignored. (As a minimum, support for textMode is expected.)

4.2.3 Styling Attributes

Styling in *aText* is handled in a manner that is generally compatible with DFXP (which was designed to be compatible with XSL 1.0), although some of the styling element and attribute names have been adapted to highlight their role in text styling only.

The following styling attributes are permitted within an *aText* specification. In all cases, the default value is underlined.

- textAlign: specifies how text is aligned in a layout area. Permitted values are:

  {start | end | left | right | center | inherit}

  (In left-to-right languages, textAlign="start" is equivalent to textAlign="left"; in right-to-left languages, the declaration textAlign="start" is equivalent to textAlign="right".)
- textBackgroundColor: specifies the background color used to fill the area around text content within the (portion of the) layout area. The permitted value is a CSS2 color specification.
- textColor: specifies the color used to render text content within the (portion of the) layout area. The permitted value is a CSS2 color specification.
- textDirection: specifies the direction of text added to a layout area based on the Unicode Bidirectional algorithm [6]. Only simple text direction control is available in *aText*; for more elaborate text processing (including tb-lr), an external text formatting language should be used. Permitted values are: {inherit | ltr | rtl}

  If no direction is defined elsewhere in the document, the effective default is "ltr".
- textFontFamily: defines the family of font used to render text. Permitted values are:

  {monospace | sansSerif | serif | inherit}

  The resolution of a generic family name to a specific font instance is not defined by *aText*, but may be defined by a specific *aText* implementation. If an unrecognized font family is specified, then monospace is used. The sansSerif and serif fonts are expected to be proportional.
- textFontSize: defines the size of the font to be used. In order to reduce implementation burden and to provide scalability across device classes, only absolute and rela-

tive sizes (as defined in XSL 1.1) are supported by *aText*. Permitted values are: {absolute-size | relative-size | inherit}
The absolute sizes supported are:
{xx-small | x-small | small | medium | large | x-large | xx-large}
The relative sizes supported are:
{larger | smaller}
As recommended by XSL, the step-factor for absolute sizes is 1.2.

- textFontStyle: defines the style of the text displayed in the (portion of the) layout area. Of the styles allowed by XSL, only the following subset is permitted in *aText*:
  { normal | italic | oblique | reverseOblique | inherit }
  If a defined style is not available, normal will be used.
- textFontWeight: defines the weight of text displayed. The values permitted in *aText* are: {normal | bold | inherit}
  If bold is not available, then normal will be used.
- textWrapOption: defines whether automatic line wrappung is supported. The values permitted in *aText* are:
  { wrap | noWrap | inherit}
  The correct wrapping of text is a processing-intensive activity that may require knowledge of the language being used. Implementations are expected to provide best-effort support for text wrapping.

There are several mechanisms for applying styles to text objects. Text styles may be applied as attributes to the top-level *aText* element:

```
<aText textColor="blue" textFontSize="large" ... >
 ... text content ...
</aText>
```

If styling is to be applied to only a portion of the child content within the *aText* element, the <span> element can be used to delineate addition styling attributes:

```
<aText textColor="blue" textFontSize="large" ... >
  ... text <span textFontWeight="bold">content </span> ...
</aText>
```

The <span> element may not contain any timing information: the <span> is only used for styling. The <span> can be made a conditional element if such a facility is supported by the host language.

In order to make the application of a common set of styles more convenient, *aText* also defines a styling container that can be placed within a document <head> section:

```
<head>
  <textStyling>
    <textStyle id="headlines"
      textColor="blue" textBackgroundColor="orange"
      textAlign="center" textFontFamily="sansSerif"
      textFontSize="large" textFontWeight="bold" />
  </textStyling>
</head>
<body>
  ...
  <aText begin="10s" dur="3s" textStyle="headlines" >
    A Geometric Tour of New York
  </aText>
  ...
</body>
```

In this example, the pre-defined style headlines could be applied to any *aText* element or used within any <span>. The aggregated text styles could also be applied to a layout container, to define the default styling for all content rendered in that container.

Of the styling attributes, the following apply only to an entire <aText> container and thus may not be overridden on via a <span>: textAlign, textDirection, textWrapOption. If text style attributes are referenced on an element to which they do not apply, they are ignored. If multiple instances of the same styling attribute are defined, the value associated with the lexically-last instance is used.

4.3 *aText* Extensions

In the current version of *aText*, each <tev> / <clear> element only may contain a single time marker that represents a single temporal moment. The use of both a start and end marker — used in many subtitle formats — is not supported in this version of *aText*: we feel that this is only useful if a <tev> is used as a fragment container, and that *this* is only useful if content can be dynamically reflowed. This is a extension of fits within the current model, but is not supported in the initial version of aText.

While the *aText* format was developed as an embedded text structuring language, it is possible to also use *aText* as an external container format. In this case, the *aText* file will contain intra-block formatting and timing control, with layout and general rendering control defined in the host language. The primary advantage of using *aText* as an external format is that the text content can be bound to the presentation at document run-time, rather than at document author time. The text can be automatically generated based on information on the presentation user, or it can be dynamically updated from a streaming source.

The current version of aText does not support structuring of text content using <div> or <p> elements. The structuring of blocks of text bring with it the need to define and customize the behavior of each element, which in turn requires a richer positioning model than was required for the seven use cases for timed text that we defined in section 2. Individual implementations may provide support for these elements.

5. IMPLEMENTATION STATUS AND PERFORMANCE ISSUES

The *aText* format described in this paper has been implemented as part of the CWI Ambulant open source SMIL player. Implementations exist for the MAC OS X version of Ambulant and additional implementations for the Nokia 800 PDA, Windows and Windows CE have been (or shortly will be) completed. An architectural definition of the integration of *aText* with the MLIF format is currently in progress.

One of the main design goals of *aText* was the development of a format that is easy to implement on a wide range of platforms, and that provides the performance level to support all of the applications classes described in section 2. The performance impact for each class is discussed briefly in the following paragraphs:

- *Headlines* and *Labels*: both of these classes require the efficient processing of short text fragments. In our experience, neither headline text or label text makes extensive use of intra-object timing, so that the main performance gain from *aText* is an increase in authoring convenience.
- *Captions/subtitles* and *Foreign-language subtitles:* both of these classes require the sub-second processing of relatively short strings of text. The exact performance gain using an embedded format depends on a number of factors, but it is clear that the use of embedded text will never require more resources than an external format.
- *Timed-constrained moving text*: the primary performance benefit for moving text is the ability to easily couple the motion rate to other elements in the host document. For example, the duration and also the pause/freeze behavior of a group of items can be most efficiently modelled with an embedded text format. This being said, the actual cost of animating the text object remain roughly equivalent in both the embedded/external cases.
- *Inter-object triggered text* and *Conditional timed text*: we see these two classes as holding the greatest potential benefit for using the *aText* format. The fact that each text fragment within an *aText* element shares a common id-namespace and a common temporal base with other objects in the presentation provides unexpected flexibility for constructing dynamic and interactive declarative presentations. The details of inter-object triggering and state-based conditional timing are beyond the scope of this article. Interested readers should consult [8].
- *Static block text*: the *aText* language allows users to define blocks of text with fine-grain styling properties. The language does not support coarser-grain styling and structuring, such as the <p> or <div> elements, or full CSS/XSL styling. The main reason for not providing such support is that a dynamic reflow algorithm can be avoided within the *aText* engine. We feel that these use cases can better be served by applying specialized external text formats.

On the whole, our initial experiences with *aText* have been very encouraging.

It is possible to take an *aText* element and to transform its content directly to the existing external formats discussed in Section 3. It is also possible to take substantial subsets of these existing formats and to transform them into *aText*. Where choices needed to be made, we generally followed the conventions used by DFXP; the existing facilities provided by RealText — considering the tight integration between RealText and SMIL — were also exploited in *aText*.

6. CONCLUSIONS

The *aText* format was designed to meet the needs of seven typical multimedia applications classes, and to provide adequate support for an eight class. Based on our initial implementation experience, we feel that this design goal has been met. An open source implementation of *aText* is available as part of the Ambulant open SMIL player and a second implementation is under development for the MLIF multi-lingual text format.

The current version of *aText* has been submitted to the W3C's Synchronized Multimedia working group for consideration as a new content module for SMIL 3.0. Based on the needs of the standardization effort, a revised version of *aText* is expected to be produced by mid-2007.

ACKNOWLEDGMENTS

This work was supported by the ITEA project Passepartout, by the NWO project BRICKS, and the IST-FP6 project SPICE. The development of the Ambulant Player is supported by the NLnet foundation. The authors wish to thank Sjoerd Mullender (CWI), Geoff Fried (WGBH/NCAM), David Kirby (BBC) and Eric Hyche (RealNetworks) for their useful comments on earlier versions of this work.

REFERENCES

[1] Adams, G., *Timed Text (TT) Authoring Format 1.0 – Distribution Format Exchange Profile (DFXP)*, W3C, 2006.
[2] Berglund, A., *Extensible Stylesheet Language (XSL) Version 1.1*, W3C, 2006.
[3] Bulterman, D.C.A., et al, *The Ambulant Open Source SMIL Player*, Proc ACM Multimedia 2005.
[4] Bulterman, D.C.A et al., *Synchronized Multimedia Integration Language (SMIL)*, W3C, 2005.
[5] Cruz-Lara, *ISO/TC 37/SC 4 N 328 rev00: MLIF*, ISO, 2006.
[6] Davis, M., The Bidirectional Algorithm, Unicode Standard Annex #9, Unicode, Inc. 2006. http://unicode.org/reports/tr9/
[7] ISO/IEC 14496-17:2006, *MPEG-4 Part 17 Timed Text Format*, 2006.
[8] Jansen, A.J., Bulterman, D.C.A. and Cesar, P.S., *Adding Document State Control for Multimedia Presentations*, Proc. ACM Multimedia 2007. (Submitted.)
[9] Lie, H.W and Bos, B., *Cascading Style Sheets (CSS)*, W3C, 1999.
[10] Microsoft, *Introduction to HTML+TIME,* 2002. http://msdn.microsoft.com/workshop/author/behaviors/time.asp
[11] Newman, D. et al., *XHTML+SMIL Profile*, W3C, 2002.
[12] Pemberton, S. et al., *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)*, W3C, 2002
[13] Raggett, D., *Hypertext Markup Language (HTML)*, W3C, 1999.
[14] RealNetworks, *The Real Networks Production Guide*, 2004, http://service.real.com/help/library/guides/ProductionGuide/prodguide/realpgd.htm
[15] Wikipedia, *SubRip: Subtitle Ripping*. http://en.wikipedia.org/wiki/SubRip
[16] 3GPP, *Timed text format (Release 6)*, TS-26245, 2004.