

Enabling Adaptive Time-based Web Applications with SMIL State

Jack Jansen and Dick C. A. Bulterman
CWI: Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam, the Netherlands
+31 20 5924300
{Jack.Jansen,Dick.Bulterman}@cwi.nl

ABSTRACT

In this paper we examine adaptive time-based web applications (or presentations). These are interactive presentations where time dictates the major structure, and that require interactivity and other dynamic adaptation. We investigate the current technologies available to create such presentations and their shortcomings, and suggest a mechanism for addressing these shortcomings. This mechanism, SMIL State, can be used to add user-defined state to declarative time-based languages such as SMIL or SVG animation, thereby enabling the author to create control flows that are difficult to realize within the temporal containment model of the host languages. In addition, SMIL State can be used as a bridging mechanism between languages, enabling easy integration of external components into the web application.

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

Design, Experimentation, Standardization, Languages.

Keywords

Declarative languages, SMIL, Multimedia web applications, Delayed ad viewing.

1. INTRODUCTION

This paper examines technology to create adaptive time-based web applications. These are applications that use time as a major structuring paradigm, and need to adapt to changes at runtime. Such adaptation can be in the form of user interaction, but also other environmental changes such as location information or a change in available bandwidth. We want these applications to be good web citizens: searchable, accessible, structured, reusable, etc.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DocEng '08, September 16-19, São Paulo, Brazil.

Copyright 2008 ACM 978-1-60558-081-4/08/09...\$5.00.

Traditionally, the web has preferred structured declarative solutions over imperative ones: HTML, CSS, SMIL, SVG and many other web standards are all mainly declarative languages. The advantage of declarative languages in a web setting is that they facilitates reuse, accessibility and device independence [11]. However, at a lower level, imperative languages (mainly JavaScript) are often required to enable time-dependent rendering, interactivity or glueing components together. This is a problem if we want to create adaptive time-based web applications, as these applications indeed require timing and interactivity and often the help of external components. The introduction of scripting into a webpage is a powerful tool, but therefore also a dangerous one: maintaining the advantages of the structured declarative model is not automatic, and may sometimes be impossible.

The alternative to structured declarative solutions is to use a technology like Flash. This is a proprietary binary format, however, and in its distribution form it has no easily parseable structure. This forestalls search and (third-party) reuse. Moreover, adaptivity and accessibility have to be explicitly catered for by the document author.

If we examine again the structured declarative languages that have an execution model (SMIL, SVG Animation) one piece of missing functionality is a user defined data model. Adding such a data model would allow a larger problem domain to be addressed without the need for a scripting language.

This paper introduces SMIL State, a technology that combines temporal web languages like SMIL or SVG with an external data model. This enables the use of free variables, thereby allowing the author to escape the temporal containment model in a controlled fashion. The data model is externalized, allowing it to be shared with other components and effectively enabling its use as an API between components of a web application.

The paper is structured as follows. In section 2 we sketch the type of applications we want to enable, and describe an example of such an application. We then outline the requirements of these applications. In section 3 we look at existing technology, and investigate how well it matches our requirements. In section 4 we describe our solution and the motivation behind it. In section 5 we report on our initial implementation. In section 6 we describe two example presentations and their architecture. We conclude with determining how well our solution matches our requirements, and some ideas about future work.

2. SCENARIO

In this paper we will concentrate on presentations which have time as their major structuring mechanism and user interaction/

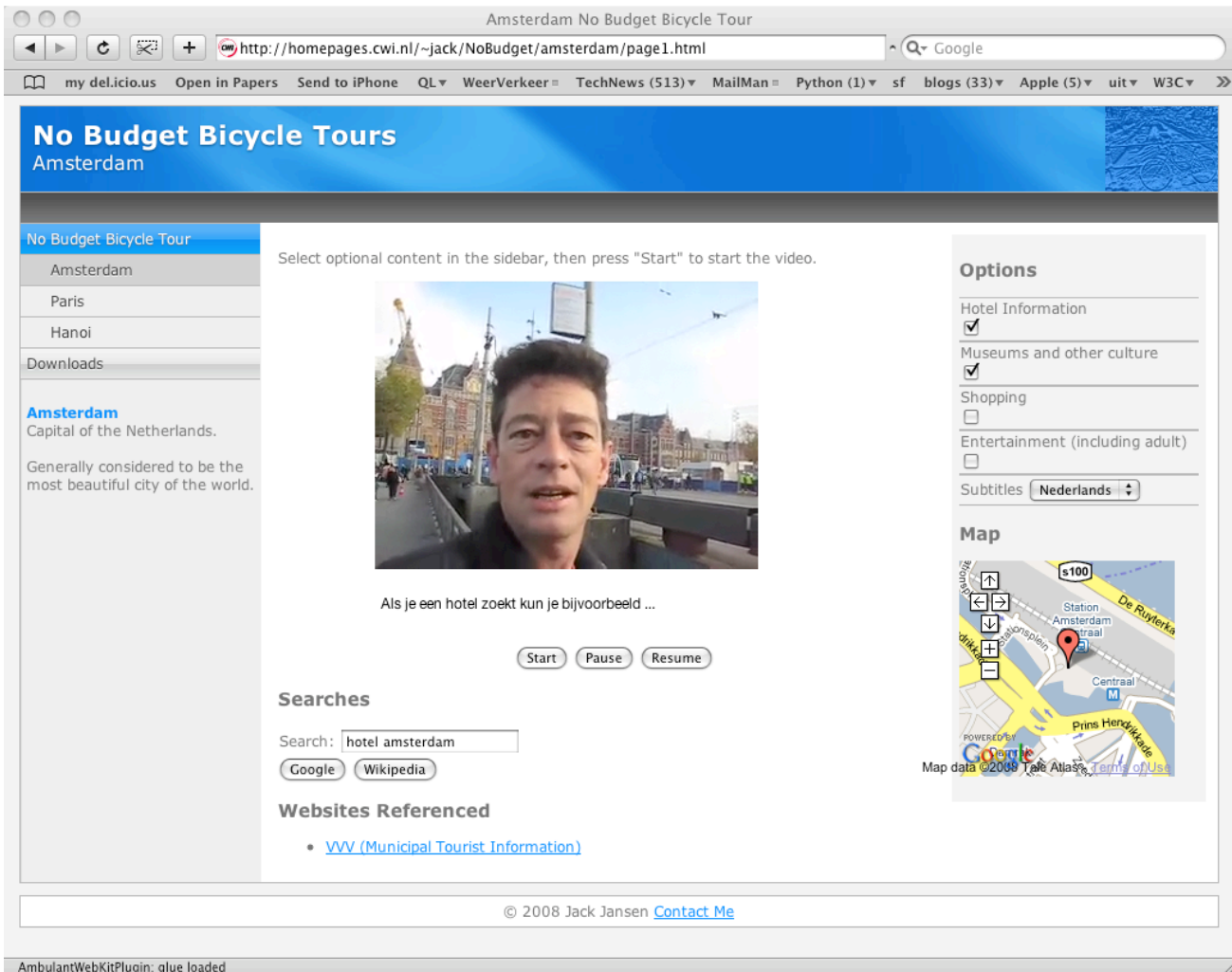


Figure 1. Screen shot of guided tour webapp

selection as the secondary mechanism. To set the stage, let us start with an example of the type of presentations we want to enable.

The application shown in figure 1 is a web-based guided tour through Amsterdam. The backbone of the application is a video, with the tour guide showing some highlights of the city, with additional information on hotels, shopping opportunities, entertainment and nightlife. The application allows viewers to select which topics they are interested in: for example, if a viewer is staying with friends and prefers to be in bed right after dinner he can choose to skip the hotel and nightlife entertainment information. The user should be able to change the content selection while viewing the presentation: if it turns out the cultural information is too detailed for his taste he should have the option to disable it on the fly.

The video presentation itself is rather fast-paced: the presenter races through the streets on his bicycle as only an Amsterdammer can and gives only terse information on the various subjects. However, for each item described the viewer is given the option of getting more information from external resources: when a museum is described the link to the museum website is also given, and the end user can temporarily pause the video to visit the museum website to find out about opening hours, etc. The

presentation also includes a standard map, such as from Google Maps, that shows where in the city the presentation currently is. This allows the end user to bookmark a place that interests her, or again pause the presentation to search for other interesting place in the vicinity. Dynamic adwords, which bring up sponsored links relevant to the material currently presented, is another possible feature.

All these are examples of the use of *timed metadata (annotations)* in the presentation. The time logic of the presentation need only know which metadata pertains to which (timed) sections of the presentation. The actual presentation of the metadata is handed off to other components for rendering.

This application is similar in scope to the personalized multimedia tourist guide described by Scherp and Boll in [16], but where they generate personalized applications on the server, our solution allows client-side personalization. This not only distributes workload from the server to the clients, but has the added advantage that viewers can adapt their preferences during playback. Another form of adaptability that we aim for is device independence: depending on characteristics of the device on which the presentation is viewed (bandwidth, screen size) some content may be replaced by items more applicable to the situation.

If this could be done dynamically, so session transfer becomes possible, that would be an advantage: transferring the presentation to another device would then only require moving the presentation over to that other device as-is, the presentation itself would adapt to the new hardware characteristics.

Another important feature for presentation authors is reusability: if a general structure can be set up that handles multiple related presentations (such as bicycle tours for other cities, in our example) that is a time-saver. It also eases serving such presentations from a content management system. A related form of reuse is third party enrichment, which requires that it is possible to refer to portions of the presentation, either in-context or out-of-context. Such reuse is increasingly important on the web, and handled well for non-temporal media through wikis and blog syndication. We want to enable this form of reuse for multimedia presentations as well.

Finally, we feel accessibility is important. Not only does this enable the use of assistive technology, but it also allows search engines to index the content inside the presentation. This is another step in enabling third party reuse: to enable someone to refer to our content they must be able to find it first.

2.1 Requirements

To enable the type of applications sketched in the previous section we have a number of requirements on the technology we use. Let us outline our major requirements, so we can then determine how applicable various technologies are to our problem space.

The solution should be *structured*. Declarative structured languages have proven themselves to be facilitate reuse, accessibility, device independence and transformability.

Time based structuring is required, because time is a major structuring paradigm for the types of applications we envision. Having time as a first class citizen allows easier presentation creation and deep-linking. Time based structuring also enables close coupling of annotations with the media fragments they refer to, ensuring they stay together in the face of edits or deep linking.

Fragment support on original media items is required. If there are multiple possible timelines through the presentation, lack of fragmenting original media would require the author to statically create multiple edits for each of the different timelines, or a large collection of small media snippets. Fragmenting support on the final presentation is also needed, again to enable third-party annotation.

Variables are required to enable presentations to adapt to user input, especially if this adaptation is to happen at a different point in time than the input itself. Variables also enable interaction patterns not foreseen by the designers of the language.

Language bridging is related to variables, but with a different scope. It is needed to enable integration of multiple components. Enabling multiple components allows the use of the best tool available for the sub-problem at hand. Language bridging and variables should also enable two-way communication between components, which increases the richness of the presentations possible.

Adaptability is needed to enable platform independence, among other things. Built-in adaptability eases the burden on the author.

Accessibility enables the use of assistive devices. Accessibility together with structuring enables search engines to index the content of the presentation.

Reusability also eases the burden on authors, by allowing parts common to multiple presentations to be implemented only once. Content management systems and other dynamic methods of creating presentations benefit from it too, as only a single instance of common items needs to be stored. Third party modification and enrichment of existing presentation also requires reusability to be feasible without copying.

3. EXISTING TECHNOLOGY

Given the requirements of the previous section we will examine existing technology. We will start with languages that aim at solving the whole problem, or at least a large subset of it. Then we will look at emerging partial solutions that may be used to augment those solutions and other related work. We will then see how well all of these match our requirements.

3.1 Multimedia on the Web

For interactive multimedia on the web there are currently two solutions in widespread use: Flash [1], and JavaScript combined with a plugin to handle media playback (such as RealPlayer or, again, Flash). SMIL, which we will examine more in-depth in the next section, is not currently a serious contender in this market.

The first solution is by far the most common, and used by websites like YouTube and Asterpix. All interaction is programmed explicitly in ActionScript, requiring specialized skills and tools. Moreover, due to the binary nature of Flash, the content is no longer easily accessible from outside. This is a problem for screen readers and other assistive technology, but also for web crawlers (content inside Flash does not show up in a search engine) and deep linking (no syndication or mashups).

Interactive multimedia presentations can also be created using standard technology: HTML, JavaScript and CSS. For audio or video playback this requires either the proposed HTML5 video extensions, or a plugin to render the continuous media. While it is usually possible to control the media playback engine from JavaScript, for example starting and stopping video playback in response to user interaction, the reverse is usually not true: having the JavaScript react to events in the video (such as specific timecodes) is not easy. In practice this means that using JavaScript is currently usually limited to presentations using predominantly static media: if time is the primary structuring paradigm of the presentation Flash is a better solution. A prime example of doing multimedia presentations with only standard technology is the W3C Slidy tool [15], which can be used to create interactive accessible slideshows.

3.2 Declarative Alternatives to Scripting

Both technologies sketched so far share the property that the logic is expressed in a procedural language (JavaScript or ActionScript). If it were possible to express the logic in a declarative way that would be more suited to the trend in web languages towards declarative structuring to enable transformability, reuse and accessibility. An example of this trend is XForms [2], which uses a wholly declarative logic to specify not only the forms themselves but also the way these forms are connected to the underlying data store. In the context of this paper we are not so much interested in the model-view-controller paradigm of XForms or the high-level definition of the controls themselves (which allows reuse of an XForm designed for a desktop web browser to be reused on a mobile browser or even a voice browser [8]). We are however interested in the declarative nature in which constraints on input values can be specified, such as “weekday must be an integer between 0 and 6 inclusive”. This feature means that old-style HTML forms that used procedural

logic in JavaScript to check value constraints can be replaced by a declarative XForm.

XForms uses an XML document as its data model, and addresses the data items in this model through XPath expressions. XForms 1.0 does not have an execution model, but it does not really need one for its application area. It does include a spreadsheet-like functional programming construct that allows variables to be computed on the basis of other variables, and that is good enough for its domain.

While it would probably be possible to create a complete interactive multimedia presentation using the technologies outlined in this section it would suffer from the fact that none of these languages have an inherent concept of time. Hence, all temporal relations would have to be explicitly coded in a language that isn't primarily meant for it.

3.3 SMIL

SMIL 2.1, the Synchronous Multimedia Integration Language [3, 4] is the W3C standard for presenting multimedia on the web. It is primarily an integration language: it contains references to media items, not the media data itself, and instructions on how those media items should be combined spatially and temporally.

SMIL is a declarative language: relations between media objects (and substructures) are described, and the computation of the timeline follows from this. The main temporal composition operators available are parallel composition, sequential composition and selection of optional content. Composition is hierarchical: nodes cannot become active unless all of their ancestors are active. The declarative containment model has one large advantage: SMIL presentations can adapt automatically to varying bandwidth conditions and alternate content with different durations. The hierarchical temporal composition model is also a nice container for timed metadata, and allows structure-based deep linking into the content.

There are a number of mechanisms in SMIL that allow the presentation to react to user input (events) and to modify the behavior of other sections of the presentation (SMIL Animation) but none of these break the basic containment model, they only modify behavior within those constraints.

The containment model has one serious drawback, though: there is no way in which the path taken through the presentation can be used to influence future behavior of the presentation. Or, more informally: there are no variables. In addition, with events being the only communication channel a SMIL presentation can not exchange structured data with the outside world. This is a problem SMIL shares with many declarative languages. For example, functional languages have had to add constructs like effect classes [6] or monads [12, 18] to enable side-effects and input/output. Without these, their application domain would have been severely limited.

3.4 Other Related Work

The technology described in sections 3.1 and 3.3 aim at addressing a large subsection of our problem space, but all have some shortcomings. In this section we will examine some ways to address those shortcomings and some solutions that address related problem areas, from which we may learn something.

XBL [7] is a language that allows an author to declaratively add behavior to an otherwise static HTML or XHTML document. It can modify the target document in-place, for example setting attributes on one element based on values obtained from another element. These actions can occur statically, somewhat similar to

how XSLT would operate on a document during load time, or dynamically, reacting to DOM events. XBL has no notion of time or control flow, so using it to create self-paced multimedia presentations would be difficult.

XConnector [13] is an extension to XLink that has some overlap with XBL in application area. It also allows the specification of relations between different elements and attributes within an XML document. Some of these relations allow similar constructs as in XBL, like changing an attribute value to match an attribute value elsewhere in the document. XConnector does have a notion of time, allowing the author to specify that something should start when something else stops, for example. The accompanying language XTemplate [14] allows an author to declare templates for such relationships, thereby enabling, among other things, the definition of temporal and spatial constraints on items in an HTML page in a way that facilitates reuse. XConnector and XTemplate together with HTML should enable creation of rich multimedia applications for the web fairly easily.

XHTML+SMIL [3] is similar to XConnector plus XTemplate, but more limited in scope (and the same is true for the current state of the work happening on SMIL Timesheets): they allow the application of SMIL timing constructs to static HTML (or other XML) documents, thereby adding timing to an otherwise static format.

Another approach is taken by King, Schmitz and Thompson in [9] (unfortunately for reference purposes, no name is given for their work, so we will call it "KST" in this paper): adding rich transformations and expressions to a language that already has an execution model, such as SMIL or SVG animation. Where SMIL and SVG animation allow only a predefined number of operations on attribute values, determined by the language designers, this paper adds spreadsheet-like expressions and conditions through a functional "little language". The temporal constraints of SMIL animation are still in place, however.

Those temporal constraints are lifted by the same authors in [17], which adds a `<value>` element that can be used to store free variables. (It also adds an templating mechanism, but that is outside the scope of this paper). This leads to a solution that has comparable application area and power as SMIL State within a single document, but the externalized data model of SMIL State allows communication with the outside world too.

3.5 Comparison

Table 1 summarizes how existing technology matches the requirements from section 2.1. The first two columns show the main problems with the most popular current solutions: a finished presentation is a monolithic unstructured blob. This results in problems for deep-linking into a presentation, but also for accessibility, which also requires access to the internals of a presentation.

SMIL 2.1 does fairly well on the structuring front, but falls short in practical issues like rich interactivity and integration with other components. Embedding XForms islands into a SMIL presentation does not help: it enables the end user to fill in forms that can be transmitted back to a server, but no extra interactivity is added. SMIL+XBL looks better, but here the generality of what XBL allows would break some of the basic assumptions of SMIL, such as timegraph consistency. Incidentally, SMIL+JavaScript, which is not in the comparison table, would have the same problem.

Table 1. Technology comparison

	Flash	Javascript +DOM	SMIL 2.1	SMIL + XForms	XBL	KST	XConnector + XTemplate	XHTML + SMIL	SMIL + SMIL State
Structured	-	-	+	+	n/a	+	+/-	+/-	+
Time based	+	-	+	+	-	+	+/-	+	+
Fragment support	-	-	+	+	n/a	+	+	+	+
Variables	+	+	-	-	+	+	+	-	+
Language Bridging	-	+	-	-	+	-	unknown	-	+
Adaptability	+/-	-	+	+	+	+	+	+	+
Accessibility	-	+/-	+	+	n/a	+	unknown	+	+
Reusability	-	-	+/-	+/-	+/-	+/-	+	+/-	+

KST is aimed at a different problem, but it still fits our requirements pretty well, with the exception of enabling communication with other components, which is outside its scope.

Interestingly enough, KST use different solutions in a number of areas where they were facing the same design decisions as us:

- both solutions allow for rich data structures in the data model, but where we opted for XML for easy sharing, they feel a richer and more compact representation is needed;
- we think static strong typing is generally not needed for most applications, and can easily be added when needed through XSchema (following the model of XForms), their solution has static strong typing;
- their solution uses an expression language based on JavaScript expressions, ours uses XPath expressions, for standards compliance.

These different choices are partially dictated by different application areas, but probably partially by personal taste as well. We agree that XPath is not a very nice language to express complex expressions in, the corresponding expression in KST is definitely more readable. XPath expressions, however, are richer in the handling of complex data structures. In the case of static typing or not this is probably more a matter of personal preference.

XConnector and XTemplate are the best fit of the existing technologies, but it shares the XBL problem that they provide so much freedom that an author has to be careful not to lose the structuring advantages of the declarative model. The same is true for temporal structuring: this can be done by an author, but the language does not enforce it. We are also not sure whether XConnector provides any help with language bridging, the literature does not mention this.

XHTML+SMIL has similar advantages and shortcomings as SMIL 2.1, which is to be expected given their common heritage.

We will explain how SMIL plus SMIL State matches the requirements in a later section.

4. DESIGN AND ARCHITECTURE

Addition of variables and communication would enable SMIL to be used in a number of application areas that are currently out of reach for it.

Courseware is an important application area for multimedia software. One of the great advantages of using computers for instructional material is that the path through the material can adapt itself to the student. This takes the form of providing more in-depth material based on user interaction, either a “tell me more” button or the answer to a question being correct or not. Courseware also benefits from the ability to interact with problem domain specific components, to enable hands-on interaction or non-standard rendering capabilities. SMIL 2.1 has no standard way to interact with external components, and no way to base decisions on user input that occurred earlier during the presentation.

Quizzes are somewhat related, but here we also want to tally results, requiring computation. Moreover, quizzes are much more fun if your personal results can be compared to those of others, requiring communication of such dynamically computed scores to some central agent.

Games are even more interactive, and require things like a ball to move in a direction determined by the mouse position when it hit a paddle, some time in the past. And a game needs more author-defined state, to determine when the aliens have all been destroyed. As with quizzes, destroying aliens becomes much more fun if your high score is transmitted to a server.

In addition, variables would allow an author to have more control over selectively rendered content. SMIL 2.1 provides *custom tests*, which allow end-user control over whether optional content is rendered or not, but the mechanism for presenting these options to the user is determined by the rendering user agent, not the author. Moreover, custom tests may be statically evaluated at presentation startup.

A separate, but related, issue with SMIL 2.1 is that it is impossible to communicate presentation state to the outside world. This problem becomes more acute once variables are added: if the SMIL presentation represents an interactive multiple-choice exam it is probably important to communicate the results to a server after the whole exam has been taken. If it represents a game we may want to keep high-scores at a central location.

A final design guideline was that the solution should be as simple as possible but be easily extensible if required for certain application areas.

4.1 SMIL State Elements

SMIL State was designed using a two-tiered approach:

- hooks in the SMIL language to enable inclusion of a data model and expression language;
- selection of a default language for the data model and expression language.

This layered approach has the advantage that if the default expression language is not the best choice for a given application it is possible to use another expression language that is more suitable without modifying the semantics on the SMIL level. The ability to use an expression language other than the default choice of XML and XPath, however, is not relevant to this paper, with the exception of the fact that it allows for extending the data model to the richer model supported by XForms.

The hooks in SMIL are:

- a `<state>` element in the head section of the document, used to declare the data model;
- an `expr` attribute that can be used on any timed element to conditionally skip the element;
- new timed elements `<setvalue>`, `<newvalue>` and `<delvalue>` which allow changing the data model;
- a head element `<submission>` and a timed element `<send>` that allow sending and receiving parts of the data model;
- an attribute value template construct, `{expression}`, that can be used in selected attributes to interpolate data model values into attribute values.

All of these hooks are modeled after existing SMIL constructs: `expr` behaves similar to system tests and custom tests, the timed elements behave like normal media items or SMIL animation elements. The attribute value template, which was modeled after the same construct in XSLT, fits in nicely with the way SMIL defines how SMIL animation and DOM access are allowed to modify attribute values in a running SMIL presentation (the so-called “sandwich model”). Attribute value templates are only allowed in attributes where they cannot modify the timegraph of a running presentation, similar to what is defined for SMIL animation.

```

<smil>
  <head>
    <state>
      <data xmlns="">
        <wantAd></wantAd>
      </data>
    </state>
  </head>
  <body>
    <seq>
      <par>
        <video src="match.mp4"/>
        
        <setvalue begin="banner.activateEvent"
          ref="wantAd" value="commercial.mp4"/>
      </par>
      <video expr="wantAd" src="{wantAd}"/>
    </seq>
  </body>
</smil>

```

Figure 2. Sample SMIL State document

For the default data model and expression language we have selected XML as the data model and XPath as the expression language. We specifically allow XPath nodeset expressions: the data model is the XML document on which XPath operates, not the XPath variable bindings. XPath variables are used as the data model in some other standards such as DISelect [10], but this data model allows only simple unstructured scalar variables. Using the XML document as the data model allows structured values such as lists and associative arrays. To allow maintaining data model consistency, updates (by a single element) are atomic, and `<setvalue>` allows copying of subtrees.

The data model XML document may be embedded inside the SMIL document, but it is logically a separate document: the XPath expressions cannot refer to random items in the SMIL document.

Figure 2 shows an example of the use of SMIL State. The data model XML document is declared in the `<state>` element in the head section, it consists of a `data` root element with one child, `wantAd`, initially empty. The data and `wantAd` elements are not part of the SMIL language, this is really a separate XML document included inline for convenience only, hence the use of the `xmlns` attribute.

When the presentation starts the `match.mp4` video starts playing. After 10 seconds, the `banner.png` image is displayed for 5 seconds. If the user clicks on this image while it is active the value of the `wantAd` element in the data model is changed to `commercial.mp4`. The `match.mp4` video continues playing until its end, whether or not the user clicks the image. After the video has finished the second video element get scheduled. Whether it plays or not depends on the `wantAd` data model item: if it is true (or non-empty and non-zero) it does play. Which video it plays depends on the value of the `wantAd` data model element, interpreted as a URL string.

4.2 SHARED DATA MODEL

The data model of SMIL State is external to the SMIL document itself. As stated in the previous section, this forestalls random changes to the SMIL document, thereby maintaining its timegraph and its structural consistency. This has the effect that we do not lose the ability to do transformation and adaptation on the document, one of the key advantages of a declarative model.

The external data model has another advantage, however: it can be shared. In its simplest form this sharing can be between runs of the same presentation: an author can create a long-running presentation that stores data when a section has been finished. A later run of the presentation can pick this up, and start the presentation at the given spot, in stead of at the beginning.

Sharing of the data model can also be applied to multiple components running at the same time. Using a shared data model as the communication paradigm between components decouples dependencies between these components: they only depend on a common understanding of the data model. This decoupling facilitates reuse, adaptability and retargeting: if a multimedia presentation wants to show locations on a map it only needs to define that it will store the location in `/location/latitude` and `/location/longitude`. The map applet can now listen for changes to these variables and modify the map view. Reuse is facilitated because another multimedia presentation only needs to be aware of this “*data model API*” to use mapping services. Same for adaptability and retargeting: if the map applet is replaced by a different one this does not affect the multimedia presentation. And

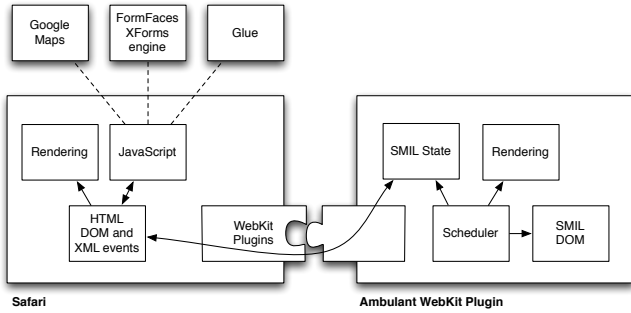


Figure 3. Browser plugin implementation

even if the map applet is completely missing, for example because the presentation is viewed on a mobile device with not enough screen space to show both the presentation and the map, the multimedia presentation need not be aware of this.

5. IMPLEMENTATION

We have implemented SMIL State in our open source Ambulant SMIL player, both in standalone mode and as a plugin for web browsers. This implementation was used to experiment with our sample applications. It turned out that the implementation of SMIL State in an existing standalone SMIL player was fairly straightforward and posed few problems.

Integration with the browser and its JavaScript implementation was more interesting, and proved more challenging. We did the initial implementation in the WebKit-based Safari browser. There was a practical reason for this: Safari exports a full API in Objective-C, Ambulant exports a full API to Python, and Python and Objective-C can be transparently bridged. All in all, 300 lines of Python code did the job of implementing SMIL State (in addition to 200 lines of Objective-C to implement the Ambulant WebKit plugin). The SMIL State design matched the platform nicely, and clean separation of components was almost automatic. The only link between the WebKit world and the Ambulant world is DOM access and XML Events, between the WebKit DOM and the Ambulant SMIL State plugin. The relevant components in this implementation are shown in figure 3.

We have also started implementing browser integration through the standard NSAPI plugin API, to facilitate using SMIL State in Firefox or IE.

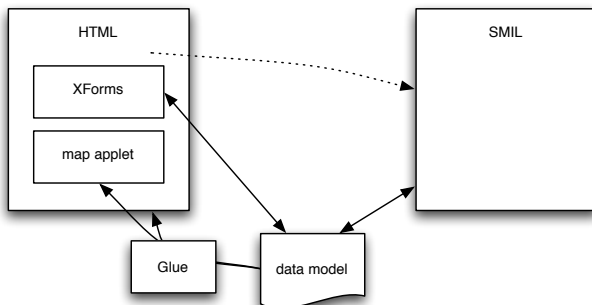


Figure 4. Guided tour document model

6. APPLICATIONS

In this section we examine two applications that address the two different aspects of using SMIL state. We start with a full-blown web app as outlined in section 2 and continue with a much more lightweight presentation that enables ad insertion into video presentation without the end-user annoyance that it currently often evokes.

These applications were created using our Ambulant SMIL playback engine, with support for SMIL State added. In case of the first application Ambulant was hosted in the Safari web browser, together with the FormFaces XForms implementation and the Google map applet. The second presentation runs in a standalone Ambulant player.

6.1 Guided Tour Webapp

We now revisit the example presentation sketched in section 2, and show how it was designed.

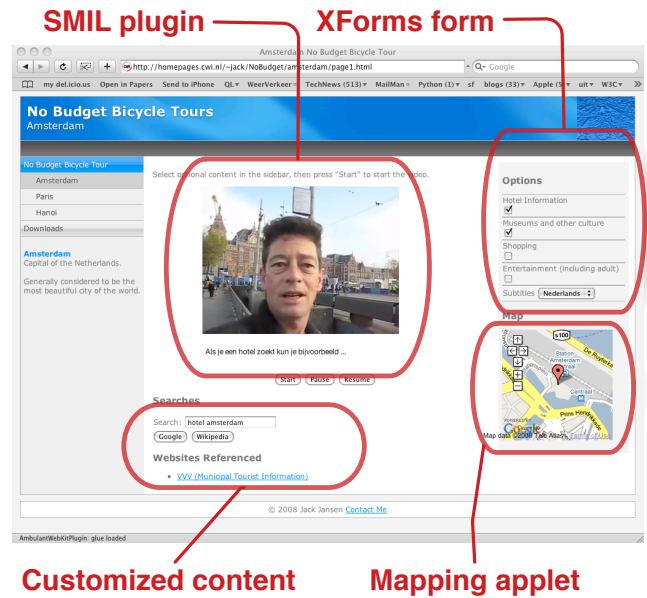


Figure 5. Components

The general control flow of the application is driven by SMIL, and consists of a linear sequence of video clips, with optional subtitles. Some clips, such as the introduction, are played unconditionally, others are played or skipped depending on user preferences set through the XForms controls. For each clip, the latitude and longitude information are stored in the data model. The location is picked up by glue on the webpage and communicated to the map applet. Additionally, references to relevant external websites, adwords and search terms are put in the data model. This information is picked up by glue code in the webpage and displayed.

Because multiple components are involved (SMIL for media playback and timing control, XForms for interaction, map applet) HTML is used as the outermost container format, as well as for displaying additional content such as background links, etc. The global structure of the presentation is shown in figure 4: the HTML document embeds the XForms form and the applet, and it

```

<head>
  <form:model ...>
    <form:instance id="jacksinstance">
      <data xmlns="">
        <optionalContent>
          <hotels>>false</hotels>
          <culture>>true</culture>
          <shopping>>false</shopping>
          <entertainment>>false</entertainment>
        </optionalContent>
        <subtitles>none</subtitles>
        <gps>
          <long></long>
          <lat></lat>
        </gps>
        <backgroundLinks/>
        <backgroundSearch/>
        <adWords/>
      </data>
    </form:instance>
    ...
  </form:model>
  ...
</head>
<body>
  ...
  <form:select
    ref="optionalContent/hotels" ...>
    <form:label>Hotel Information
    </form:label>
    <form:item>
      <form:label></form:label>
      <form:value>true</form:value>
    </form:item>
  </form:select>
  ...
</body>

```

Figure 6. HTML and XForms code

has a reference to the SMIL presentation. SMIL (through SMIL State) and XForms both refer to the shared data model, and can both read and modify it. The map applet and HTML page itself only read values from the data model, through a bit of glue. How this architecture matches to the visual representation on the web page is shown in figure 5.

This glue needs a bit more explanation: as only XForms and SMIL have direct access to the data model, in the prototype the glue is implemented with a bit of Javascript, triggered by DOM events when the data model changes. This glue could be implemented using XBL, XConnector or another declarative form, but unfortunately none of these were available in a browser that could also host our SMIL plugin.

Figures 6 and 7 show the relevant parts of the HTML and SMIL documents, respectively. The HTML document has the embedded data model (in the XForms namespace). It consists of sections `optionalContent`, for content selection, `subtitles`, for subtitle selection, and `gps`, `backgroundLinks`, `backgroundSearch` and `adWords`, for communicating timed metadata. The XForms form enables the viewer to select, for example, whether to display the hotel information or not.

```

<par expr="optionalContent/hotels">
  <video src="biketour.mp4"
    clipBegin="26s" clipEnd="53s" .../>
  <smilText expr="subtitles = 'nl'" ...>
    Als je een hotel zoekt kun je
    bijvoorbeeld ...
  <clear begin="6s"/>
  ...
</smilText>
<setvalue ref="gps/long"
  value="52.3776"/>
<setvalue ref="gps/lat"
  value="4.89868"/>
<setvalue ref="adWords"
  value="'hotel amsterdam'"/>
<newvalue ref="backgroundSearch"
  name="hotel"
  value="'hotel amsterdam'"/>
  ...
</par>

```

Figure 7. SMIL code

In the SMIL code, the whole section is played only if `optionalContent/hotels` is true. The multimedia data for that section consists of a subsection of the video clip and some subtitles. The metadata is stored in the data model at the time the media start. Some of this metadata is scalar (such as longitude, latitude and `adWords`), some is structured (background search items). In the latter case a new sub-item named `hotel` is added to the `backgroundSearch` container.

Note that, despite the similarity to SMIL Animation constructs like `<set>`, these `<setvalue>` and `<newvalue>` elements are not automatically reverted when their timeline ends. In that way, they form the procedural escape hatch for the temporal containment model, while still keeping that containment model intact in the general case.

6.2 Delayed Ad Selection

The standard way to do advertisements in video streams, whether over the internet or through traditional channels, is ad insertion. This can be static or dynamic, but the dynamism is generally server-based: depending on data the server knows it selects specific ads to insert. This selection process may be based on a user profile the server keeps, but there is no direct user interaction. Ad insertion done dynamically at client side, based on user interaction, such as discussed in [5], has a different problem: it hinges on the fact that the viewer is so interested in the product that she actually clicks the link, disrupting her viewing experience. We feel this may be a bit of a long shot.

For static media on the internet the situation is wholly different. Inserted advertisements, which require the user to first read the ad before being able to get at the content they are looking for, are generally frowned upon, and nowadays most major browsers contain features that actively try to forestall popups and other disruptive advertisements to show up. In stead of the forced consumption of ads webpages tend to work with the voluntary model: the user has the option of clicking a banner ad. While some people think even this goes too far the model has a much larger acceptance than forced ads.

For multimedia it would be good to transport the voluntary banner ad method to the realm of multimedia. However, if the user is in



Figure 8. Video with delayed ad banner

the mindset of watching a video she is unlikely to click an advertisement to watch that in stead.

To cure this we have come up with a technique we call *delayed ad viewing*. A video program has pre-determined advertisement slots, and during such a slot an advertisement always plays. However, through interaction with the presentation before the advertisement slot the user can influence which ads will be played.

The sample presentation consists of a (non-live) football program. Included in the presentation are a number of commercial videos, with a default playout order. At various times, usually when a billboard is in plain view in the video footage, a banner for a specific brand will show up in the lower-right corner of the screen for a couple of seconds. Figure 8 shows how this looks during playback. If the user clicks during this period the corresponding ad will be moved to the front of the playout list. When it is time for a commercial break the main video is paused and the head of the advertisement playout list is shown. after an advertisement has been viewed its banner will no longer show.

Figure 9 shows the timelines of three different playbacks of the same document. User 1 did nothing and got the default ad playout order of a soap advertisement and a beer advertisement. User 2 clicked the “Ford” banner, and got that advertisement first followed by the default soap ad. User 3 requested the Amstel and

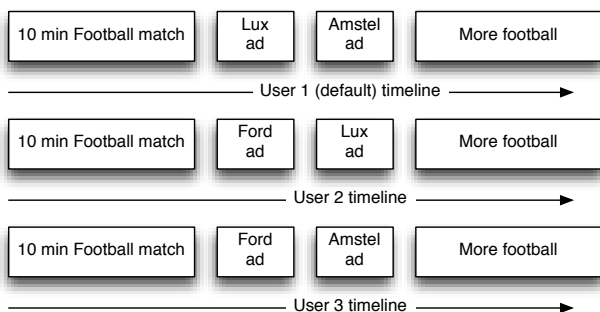


Figure 9. Different playout orders

Ford banners, and was spared the Lux ad. At least, during the first commercial break.

At the end of the presentation the state variables contain information on which ads have been watched. This information could be transmitted back to a central server for monetization, along the lines of pay-per-click ads on static webpages. Alternatively, this data could be gathered by the media server when the request to serve the ad stream comes in.

Note that the use of delayed ad selection does not preclude other current standard ad-insertion methods. The SMIL presentation can be generated on demand by the server for a specific user. Whether the user has complete freedom to select advertisements or only limited options is a choice at the discretion of the content provider. Different advertisement selections, choices and commercial break frequencies can be served to different users by serving only different SMIL documents: the underlying media items can all be static.

The structure of the presentation is rather simple, and figure 2 gives the general idea. A problem that was encountered is that the XPath expression language is primarily meant for manipulating general XML documents and not for the more spreadsheet-like operations we are using it for. Hence, functions like `max()`, which would have made the ad reordering a lot simpler, are missing and the logic needs to be written out.

7. CONCLUSIONS AND FUTURE WORK

Based on the example applications we have created we can match our solution to the requirements (see table 1). SMIL State does not interfere with any of the advantages of SMIL 2.1, so we only need to look at the three requirements where SMIL 2.1 is lacking.

Variable support works nicely in SMIL State, and simple use cases have simple solutions. XPath as the expression language could have used a little boost, though, as XForms did by introducing a number of convenience functions into the XPath function namespace.

Language bridging works fine. Here the problem is on the other side of the bridge: as only SMIL State and XForms currently share this data model, the integration into other languages requires some glue code.

Reusability works fine. Whether you want to replace components or refer to fragments inside the presentation, we have not encountered any problems.

SMIL State has been proposed to the SYMM working group, and has been accepted as a part of the standard for SMIL 3.0, which is in Candidate Recommendation status as of this writing.

We intend to pursue and extend this model in the context of the W3C Rich Web Application Backplane Incubator Group. There we will also try and address the shortcomings sketched in this section.

8. ACKNOWLEDGEMENTS

The work reported in this paper has benefited from suggestions offered by members of the W3C backplane activity and members of the W3C Synchronized Multimedia working group. Sjoerd Mullender, Julien Quint and Daniel Weck have provided comments on earlier versions of this research. We are grateful to Steven Pemberton for introducing us to the philosophy behind XForms, which seeded the design of our solution. This work has been funded by the NWO BRICKS PDC3 project, and by the FP7 IST project TA2. Development of the open source Ambulant

Player and CWI's participation in the SMIL standardization effort have been funded by the NLnet foundation. We gratefully acknowledge this support.

9. REFERENCES

- [1] <http://www.macromedia.com/software/flash/about/>
- [2] Boyer, J. 2007 XForms 1.0 (Third Edition). W3C. URL=<http://www.w3.org/TR/xforms/>.
- [3] Bulterman, D. and Rutledge, L. 2004 SMIL 2.0: Interactive Multimedia for Web and Mobile Devices. Springer-Verlag, Heidelberg, Germany, ISBN: 3-540-20234-X.
- [4] Bulterman, D. et al. 2005 Synchronized Multimedia Integration Language (SMIL 2.1). W3C. URL=<http://www.w3.org/TR/SMIL/>.
- [5] Costa, R. et al. 2006 Live editing of hypermedia documents. DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering. ACM, New York, NY, 165-172. DOI=<http://doi.acm.org/10.1145/1166160.1166202>.
- [6] Gifford, D. and Lucassen, J. 1986 Integrating functional and imperative programming. ACM conference on LISP and functional programming. DOI=<http://doi.acm.org/10.1145/319838.319848>.
- [7] Hickson, I. 2007 XML Binding Language (XBL) 2.0. W3C. URL=<http://www.w3.org/TR/xbl/>.
- [8] Honkala, M. and Pohja, M. 2006 Multimodal interaction with xforms. ICWE '06: Proceedings of the 6th international conference on Web engineering. ACM, New York, NY, 201-208. DOI=<http://doi.acm.org/10.1145/1145581.1145624>.
- [9] King, P., Schmitz, P. and Thompson, S. 2004 Behavioral reactivity and real time programming in XML: functional programming meets SMIL animation. DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering (2004). DOI=<http://doi.acm.org/10.1145/1030397.1030411>.
- [10] Lewis, R. et al. 2007 Content Selection for Device Independence (DISelect) 1.0. W3C. URL=<http://www.w3.org/TR/cselection/>.
- [11] Lie, H. and Saarela, J. 1999 Multipurpose Web publishing using HTML, XML, and CSS. Communications of the ACM, Vol. 42, Issue 10. ACM, New York, NY, 95-101. DOI=<http://doi.acm.org/10.1145/317665.317681>.
- [12] Moggi, E. 1988 Computational Lambda-calculus and monads. In proceedings 4th Annual Symposium on Logic in Computer Science. IEEE Computer Society Press, Washington, DC.
- [13] Muchaluaat-Saade, D., Rodrigues, R. and Soares, L. 2002 XConnector: extending XLink to provide multimedia synchronization. Proceedings of the 2002 ACM symposium on Document Engineering. ACM, New York, NY, USA. DOI=<http://doi.acm.org/10.1145/585058.585069>.
- [14] Muchaluaat-Saade, D. and Soares, L. 2003 XConnector and XTemplate: improving the expressiveness and reuse in web authoring languages. The New Review of Hypermedia and Multimedia. Taylor&Francis, Bristol, PA, USA. DOI=<http://dx.doi.org/10.1080/13614560208914739>.
- [15] Raggett, D. 2006 Slidy - a web based alternative to Microsoft PowerPoint. XTech (Amsterdam, May 16-19 2006). URL=<http://www.w3.org/2006/05/Slidy-XTech/slidy-xtech06-dsr.pdf>.
- [16] Scherp, A. and Boll, S. 2004 Generic support for personalized mobile multimedia tourist applications. MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia (2004). DOI=<http://doi.acm.org/10.1145/1027527.1027566>.
- [17] Thompson, S., King, P. and Schmitz, P. 2007 Declarative extensions of XML languages. DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering (2007). DOI=<http://doi.acm.org/10.1145/1284420.1284442>.
- [18] Wadler, P. 1990 Comprehending Monads. In Proceedings of the 1990 ACM Conference on Lisp and Functional Programming, pages 61--77, Nice, France, 1990.