Adding State to Declarative Languages to Enable Web Applications

Jack Jansen
Vrije Universiteit, Amsterdam

Adding State to Declarative Languages to Enable Web Applications

Jack Jansen
Vrije Universiteit, Amsterdam
Student number 0197254
Jack.Jansen@cwi.nl

Supervisor: Prof. dr. D.C.A. Bulterman

Abstract

On the web, media tend to be encoded in declarative formats, which facilitate accessibility, reuse, and transformation. Web applications, on the other hand, are created with more procedural technology and do not enjoy these benefits. In this thesis we examine how this can be fixed. We examine a small part of the problem space, *adaptive time based applications*, and investigate how we can extend existing declarative languages to facilitate these. We develop a mechanism, *SMIL State*, which enables SMIL to be used to create such applications, either by itself or integrated with other declarative components. The mechanism is then evaluated in its target application area. We then return to the larger problem area, and show that there are opportunities for applying similar techniques. This should eventually enable the creation of web applications that are more integrated in the web as a whole, by being searchable, accessible, transformable and reusable.

Table of Contents

1. Introduction

The World Wide Web was conceived in the early 1990's with information sharing as its primary goal. Initially, the web was predominantly a text-based medium, with support for hyperlinks and simple bitmapped graphics. Over the years many other languages were developed to allow a richer experience to delivered to the client: CSS decoupled presentation aspects from semantic meaning, SVG enabled structured graphics (as opposed to bitmapped graphics) and MATHML allowed mathematical formulas to be represented, to name just three. These languages are structured declarative *Domain Specific Languages* (DSL's): they are designed for a specific problem domain, not general purpose computation, and they emphasize the *what* over the *how*.

The declarative nature of web languages has a number of key benefits:

- The declarative design enables web technology to *adapt* well to differences in many aspects of the environment. These differences in the environment can be of a technical nature, such as bandwidth and screen space, but also of a personal nature, such as the end users' mother tongue or specific disabilities. The use of declarative technology enables document authors to focus on the content, while the mechanisms needed for the adaptation are handled by the implementation. For example, an XForms single selection control is usually rendered as a drop-down menu, but if screen space is limited it may be a rollover menu, and a voice browser could read out the entries.
- The text-based structure enables *accessibility*: a blind mathematician is much better served by a MATHML fragment read by a speech synthesizer than by a bitmapped graphic of that same formula. In addition, accessibility promotes *searchability*: search engines can index text more easily, and the structure of documents enables them to present results in context.
- The structured nature facilitates *reuse*. This is not only reuse by the original author, but structure combined with hyperlinks also enables third parties to refer to relevant items in a new context (*deep-linking*).

These benefits, combined with the fact that these formats are open standards, have played a large part in shaping the web as we know it today: accessible from different devices, easily searchable, etc.

One area of the web where the declarative model has not been widely used is that of active content. In parallel to the enrichment of media formats, development was happening on how computation was handled. Initially, any computations required were done on the server side, with the results delivered to the browser in HTML or other static representations. By the middle of the decade, however, various technologies appeared that enabled computation to be run client side, in the browser. Over the years, JavaScript, Flash and Java became the most popular of these, due to two factors:

- *Machine independence*, due to being based on a virtual machine, and
- *near-ubiquitous availability* in browsers, allowing end users to view the content without first having to install a new plugin.

Initially used for simple scripting-like tasks such as animated buttons, over the years *Web Applications* (webapps) started to appear: applications with a similar role as traditional desktop applications, but delivered over the web and running in the browser. Eventually, the richness of the web as a source of data and the ability to aggregate input from users around the globe has enabled a whole new category of applications.

Unfortunately the two trends sketched here have proceeded largely independent: the DSL's used for static content did not provide enough freedom of expression for the creation of Web Applications, so the trend has been to use JavaScript. Development of the field has taken the form of standardizing JavaScript, and extending it with technologies like DOM, XML Events and Ajax to enable richer applications. This has led to a situation where an advance in one area (more dynamism on the client side) has led to drawbacks in another: the easy adaptivity of using a declarative language was lost.

Currently, the benefits of the declarative model that static content enjoys is not available to web applications: accessibility (and searchability) has to be explicitly catered for by the author, there is little support for automatic adaption and reuse.

In this thesis, we will study ways to address this situation. The work presented here was done under the supervision of Prof. dr. Dick Bulterman, in the SEN5 group of Centrum Wiskunde & Informatica (CWI), Amsterdam. The research focus of the group is *Distributed Multimedia Languages and Infrastructures*, and this is reflected by work in a number of areas[1]:

- *composition languages* for multimedia, enabling media streams from different sources to be combined and allowing adaptation and interoperability between mobile, home and other platforms;
- *structured authoring*, allowing authors to focus on the story line in stead of delivery or packaging details;
- *end-user authoring*, enabling viewers to enrich and share content from a home setting, looking both at technology such as secondary screen usage as well as social issues (sharing, privacy, etc).

---

1 The current list of publications of the group can be viewed at http://repository.cwi.nl:8888/cwi_repository/zoekinoaienora/searchrepository.php?stheme=SEN5 .

- *infrastructure* for efficient and timely media delivery, such as playback scheduling algorithms, distributed clock synchronization protocols and priority-based routing.

The group combines a strong presence in the scientific field with a practical side: various group members are active in standardization committees, such as the Synchronous Multimedia Working Group (SYMM WG) of W3C. The group also has a history of producing open source software distributions, such as of the Ambulant Player.

1.1 Overview of the Problem Domain

The problem domain can be summarized as follows: *can web applications enjoy the same benefits that declarative languages give the static web?*

Most declarative web languages (HTML, CSS, SVG, SMIL, VOICEXML, RDF, to name a few) are examples of DSL's: they have been designed for a specific purpose. Therefore, they perform poorly (if at all) as general purpose programming languages. The ability to combine languages would help here, but unfortunately this is difficult. The languages often each have their own paradigm, and these paradigms are difficult to unify.

As an example, SMIL is about time and uses structure to represent structured timelines, which allows creation of documents that adapt to the environment without breaking the story line intended by the author. SVG uses structure to represent hierarchical objects and grouping, thereby enabling reuse of objects within a document. Combining these two paradigms in a single document can be done in two ways:

- *Using SVG hierarchy as the outer structure*. This has been done for the way SMIL Animation is embedded in SVG, which loses a lot of SMIL functionality, such as automatic adaptive timing.
- *Using SMIL timing as the outer structure*. This would lose SVG functionality, such as object reuse.

Enabling web applications to use these languages to solve sub-problems that fit the domain would be beneficial. As direct integration of the current languages is not an option the next best solution is interoperability: allow the languages to be used each in their own problem domain, while communicating with each other. Some languages interoperate well because they were specifically designed to do so, for example HTML and CSS. Some other combinations have sufficiently orthogonal domains that they have been made to interoperate through simple *syntactic sugar*, for example HTML and RDF through microformats.

Due to its history, the one language that has reasonable interfaces to most web languages is JavaScript. Through DOM and XML Events JavaScript code can control and modify code in other languages, and events happening in an-

other language can trigger JavaScript. Most web languages provide a very rich DOM model. This appears to be convenient for JavaScript programmers, but has a serious *safety implication*: the semantics of the target language can no longer be guaranteed if any language construct can be modified on the fly in an uncontrolled fashion. This results in the declarative benefits mentioned earlier becoming unsafe. The DOM interaction model is also unidirectional: JavaScript is in control, and the other technologies are driven by it. This leads to a model where the declarative constructs may be embedded in JavaScript code to create and control them, effectively abolishing the declarative advantages.

By identifying the areas where declarative web languages could interoperate to enable solutions that would not be feasible in a single language it should become possible to create more and more applications in a declarative manner.

1.2 Problem Statement

This master thesis investigates combining multiple incompatible paradigms currently prevalent on the web in a way that allows the whole to be more than the sum of the parts. The paradigms we have selected are the *structured time paradigm* of SMIL and the *data model paradigm* of XForms. This combination was selected because it is non-trivial: the two paradigms are concerned with *time* and *state*, and these two concepts interact and mutually influence one another.

The strong points of SMIL have been mentioned before: the ability to create timed documents that adapt dynamically to a changing environment while maintaining story line integrity. However, SMIL has two weak points from a web applications point of view:

- Support for non-temporal adaptation is weak: there are limited constructs that allow a presentation to react to things that happened in the past. Or, colloquially: *SMIL has no variables*.
- Integration with the environment is limited: SMIL has no way to trigger actions in other components, and its DOM model also allows other components only rudimentary control over the presentation. Colloquially: *SMIL is an island*.

XForms was designed to replace the rather ad-hoc method of data submission provided by HTML Forms with a more elegant method. It uses the model-view-controller paradigm to separate the data model from the interaction model. Because the data model is a separate XML document it integrates easily with its environment. This is true both in a client-server setting, where the data model can easily be communicated with a server, or in a web application, where JavaScript code can use DOM access and XML Events on the data

model. However, XForms has only a limited execution model (based on XML Events) and no temporal model.

In the rest of this thesis, we study how incorporation of an XForms-derived data model into SMIL can be designed, and whether it indeed enables declarative web applications. While SMIL and XForms are the focus in this work, the method is expected to have wider applicability:

- There are a number of languages with an execution model, such as VoiceXML and SCXML, which may also be amenable to a similar approach.
- Coupling modules through a data model leads to a paradigm of *data model as API*. This appears to be a promising model for loosely coupled web applications.
- The data model is completely accessible through the DOM, thereby bridging SMIL to JavaScript, which enables interaction between SMIL and most other web languages.

We hope to show that our method of combining two declarative languages through a small bridging technology provides insights that can be applied to other areas in the problem domain.

1.3 Research Questions

The following list defines the research questions that motivated the work described in this thesis:

1. Can a mechanism be designed that provides variables in a declarative language, without losing the advantages of the declarative model? In other words: does the mechanism have an advantage over simply opening the whole language to modification through DOM access and JavaScript?
2. Does the mechanism actually enable interaction between multiple languages, does it live up to the expectations? Is the mechanism rich enough to allow actual applications to be created?
3. Is there an advantage to using the new mechanism over using existing technology for writing such applications?
4. Is the mechanism actually applicable more widely than just to SMIL?

As we will see, the results obtained by this work have largely provided positive answers to the questions in this list.

1.4 Methodology

The first research question, above, is the question that originally started this research: how can we add variables to SMIL, to enable applications such as distance learning and games. We started by researching ways in which other non-

procedural languages, and specifically XML-based web languages, handle this issue. A promising approach was found in the way the XForms data model works, combining architectural simplicity with expressive power.

The XForms data model architecture was adapted to operate in a SMIL setting, with timeline driven activation (as opposed to the event-based and dependency-driven model in XForms). Evaluation of the architecture showed opportunities for wider applicability, which prompted research into applying it to the larger scope that is the subject of this thesis.

An initial strawman implementation was created in the Ambulant SMIL player, and some test documents were created to gain some experience. During this process, the solution was presented to the W3C SYMM group for feedback. Based on this feedback the design was refined, and a complete implementation was created, again in the Ambulant player.

The scientific relevance was then tested by submitting a paper entitled *Enabling Adaptive Time-based Web Applications with SMIL State* to Document Engineering 2008 (September 16-19, Sao Paulo, Brazil). It was accepted and awarded the best paper award. Subsequently, the paper was extended, and accepted for publication in *Multimedia Tools and Applications* (Springer Science and Business Media), as *SMIL State: An Architecture and Implementation for Adaptive Time-Based Web Applications*. That extended paper forms chapter 2 of this thesis..

The interaction with the SYMM Working Group resulted in SMIL State being accepted for inclusion in the SMIL 3.0 Recommendation in December 2008. The relevant chapter of the specification is included here as Appendix A.

1.5 Outline

As most of the results of this research have been previously published elsewhere they have been included verbatim.

Chapter 2 contains the MTAP paper, *SMIL State: An Architecture and Implementation for Adaptive Time-Based Web Applications*. It examines the problem area from an application point of view, from requirement analysis via design to evaluation. Section 2.1 introduces *adaptive time-based web applications* and how these can be created with current technology. Section 2.2 elaborates on those applications and examines an example of such an application, leading to a set of requirements. The application is a video-driven interactive guided tour through Amsterdam, with adaptive selection of content and interaction with map components, background material, etc. Section 2.3 investigates existing technology, related research and how well these match the requirements. Section 2.4 describes the SMIL State design and its motivation. Section 2.5 describes the implementation of SMIL State. Section 2.6 then returns to the actual applications, and examines how two of these have been implemented

with SMIL State: the guided tour application and another application that allows a novel method of advertisements in video content. This method, *delayed ad viewing*, enables an alternative to fixed commercial breaks (as in broadcast TV) and popup or click-to-view ads (as on the web). Section 2.7 has conclusions, which are summarized below.

Appendix A contains the SMIL State chapter of the SMIL 3.0 Recommendation. It contains the complete technical description of SMIL State, and the rationale for standardizing it. SMIL is a modularized standard, to enable

Appendix B contains the portions of the test suite that are relevant to SMIL State.

1.6 Conclusions

The conclusions of our work are summarized in section 2.7. In short, these conclusions are that adding a data model to SMIL does not interfere with the advantages of the declarative nature of SMIL, that it enables new, richer applications, and that such a model is easy to implement.

We can also provide answers here to the research questions posed earlier:

1. *Can a mechanism be designed that provides variables in a declarative language, without losing the advantages of the declarative model?*

   SMIL has been specifically designed to enable it to retain its declarative advantages, even in the light of DOM access and other modification, through the so-called "*sandwich model*". This is done by limiting the types of modification that a script or animation can cause. This model has proven itself over the last decade. As discussed in section 2.4, the SMIL State design fits seamlessly into the sandwich model, therefore it does not cause any new issues with the declarative model and its advantages.

2. *Does the mechanism actually enable interaction between multiple languages?*

   The MTAP paper describes one such application, the *No Budget Bicycle Tour*. This application was implemented using SMIL, XForms, HTML and widgets written in JavaScript, all tied together using SMIL State. Integration between XForms and SMIL worked seamlessly. It was expected that we could use XBL for integration with JavaScript and HTML, but due to its unavailability on our experimental platform a small amount of JavaScript glue code was required. A detailed description can be found in section 2.6.

3. *Is there an advantage to using the new mechanism over using existing technology for writing such applications?*

   The total amount of code in the bicycle tour application is small (less than 500 lines, counting all HTML, SMIL, XForms and JavaS-

cript), therefore we can conclude that a similar application created with traditional means would require much more effort. However, we have not actually conducted the experiment of re-creating the application in JavaScript.

4. *Is the mechanism actually applicable more widely than just to SMIL?*

   The best paper award for the Document Engineering paper suggests wider applicability than just SMIL. In addition, reactions have been favorable in discussions in the Rich Web Applications Backbone XG (RWAB XG), which is comprised of experts in this field. Specifically, the *data model as API* has generated sufficient interest to prompt further research.

All in all, the work presented shows that judicious application of technology from one declarative web language to another is possible, and that this can be used to enable new applications. It also appears that this method can be applied to more languages, with the eventual goal of enlarging the category of web applications that can be created in a declarative way.

1.7 Discussion

The work presented here here has a practical as well as a scientific side: SMIL State was proposed to the SYMM working group for inclusion in SMIL 3.0, and accepted in the Recommendation. The relevant section of the SMIL Specification is included as Appendix A. In the standardization process, the members of the SYMM working group examined the technology, and an independent implementation of SMIL State was created by an external party, based on the description in the recommendation. This second implementation also passed the SMIL State test suite (included here in Appendix B). In addition, the recommendation was endorsed by the whole W3C membership in the two rounds of review leading to the acceptance of the recommendation.

The SMIL DOM access model is much more limited in the modifications it allows to the DOM tree than most other XML languages. This results in DOM access not compromising the SMIL semantics, thereby retaining the declarative advantages, at the cost of limitations in dynamism. Introduction of variables through SMIL State lifts some of those limitations while still safeguarding the SMIL semantics. A parallel can be drawn to the ban on self-modifying code in general purpose programming languages and operating systems: limiting access patterns to a well-defined and understood subset gains a lot of safety while losing only a little freedom. It seems likely that a similar approach can be taken for other web languages, especially languages that have an execution model (such as SVG, SCXML or VoiceXML).

The SMIL timing and synchronization model may have wider applicability than only multimedia. While this has always been understood by the original

creators of the language, its practical application has not been pursued. One of the reasons for this has been the lack of variables, which means that the expressivity of the language has been on the level of a state machine. The introduction of SMIL State changes this and may enable the use of SMIL as a more general declarative control language. One area of interest is the XForms switch module: a set of constructs that allow tabbed dialogs or wizard-based dialogs to be expressed. XForms 1.0 standardizes a set of constructs for this that have little or no semantic relation to the rest of the language. SMIL has a much richer set of sequencing constructs here, and SMIL State would allow integration with the rest of XForms.

The "*data model as API*", introduced in section 2.4.2, is worth further investigation. It enables web applications to be created in a loosely coupled fashion, thereby not only component reuse (discussed in section 2.2) but maybe also distribution and fault tolerance. I have started some preliminary work on the former, in the form of a literature study, available at http://homepages.cwi.nl/~jack/presentations/lit-state.pdf .

Finally, in a completely different area, section 2.6.2 introduces the concept of *delayed ad viewing*, a way to monetize multimedia playback that attempts to address the annoyance of end users being forced to watch advertisements selected by the content producer. It may be worthwhile to investigate whether this model can indeed gain end user acceptance, and what the consequences are for business models, etc.

1.8 Acknowledgements

I would like to thank my supervisor, Prof. dr. Dick Bulterman, for enabling and seeding the work in this thesis. His insistence that there must be a better way to enable interaction in web languages than simply opening up everything to script-based modification is the core of the ideas presented here, and our practicality versus purity discussions have led to a solution that, in my opinion, sits in the sweet spot. Dick has also been very helpful with showing me how document structure and choice of prose can make the difference in communicating the underlying ideas.

I am grateful to Prof. dr. Guus Schreiber (VU) for agreeing to be the second reader for this thesis and for his valuable feedback, especially on all of chapter 1 and on the general structure.

The SEN5 group at CWI, especially Dr. Pablo Cesar provided the setting for this work, and gave feedback on the design and applicability.

Steven Pemberton of W3C introduced me to the XForms data model and its applicability to the problem.

The W3C SYMM WG, especially Sjoerd Mullender, Julien Quint and Daniel Weck, provided comments on the integration of the model into SMIL. In addition, Sjoerd provided a second implementation.

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.

2. SMIL State: An Architecture and Implementation for Adaptive Time-Based Web Applications

In this paper we examine adaptive time-based web applications (or presentations). These are interactive presentations where time dictates which parts of the application are presented (providing the major structuring paradigm), 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. Finally, SMIL State enables richer expressions for content control. This paper defines SMIL State in terms of an introductory example, followed by a detailed specification of the State model. Next, the implementation of this model is discussed. We conclude with a set of potential use cases, including dynamic content adaptation and delayed insertion of custom content such as advertisements.

A version of this section will appear in *Multimedia Tools and Applications* (Springer Science and Business Media) in 2009, as "*SMIL State: An Architecture and Implementation for Adaptive Time-Based Web Applications*" (authors Jack Jansen and Dick Bulterman), DOI: 10.1007/s11042-009-0270-3.

2.1 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-based information or a change in available bandwidth. In addition to being adaptive (or responsive), these applications should also be good web citizens: they (and the adaptation strategy) should be searchable, accessible, structured, reusable, etc.

Traditionally, the web has preferred structured declarative solutions over imperative ones: HTML [25], CSS [5], SMIL [7, 8], SVG [12] and many other web standards are all mainly declarative languages. The advantage of declarative languages in a web setting is that they facilitate reuse, accessibility and device independence [20]. However, at a lower level, imperative languages (mainly JavaScript [13]) are often required to enable time-dependent rendering, inter-

activity or binding of specific components. This presents 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 an imperative technology such as Flash [2]. Flash is an example of a proprietary binary format, which uses a content encoding that is – in its distribution format – difficult to parse at activation time. This forestalls search and (third-party) reuse. Moreover, any presentation and document adaptation and conditional accessibility need to be planned and explicitly catered for by the document author.

This is not to imply that declarative language already solve all interaction problems. If we examine the structured declarative languages that have an execution model (SMIL, SVG Animation), one piece of missing functionality is a user defined *data model*. A data model defines a document-specific collection of variables and settable parameters. Adding such a data model, while not completely eliminating the need for scripting, would allow a larger problem domain to be addressed without the need for a scripting language. This can make declarative documents more useful, especially in situations when document need to be generated automatically.

This paper introduces *SMIL State*, a technology that combines temporal web languages like SMIL or SVG with an external data model. SMIL State enables the use of free variables in declarative presentations, 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.

This paper is an extended version of [18], which was presented at ACM Document Engineering 2008. It widens the scope of the former paper by examining how SMIL State is applicable to enriching existing SMIL content control mechanisms and by providing more detail on the implementation and the lessons learned from that implementation.

The paper is structured as follows. In section 2 we sketch the types of applications that are relevant for SMIL State, and describe an example of such an application in detail. We then outline the requirements of these applications. In section 3,

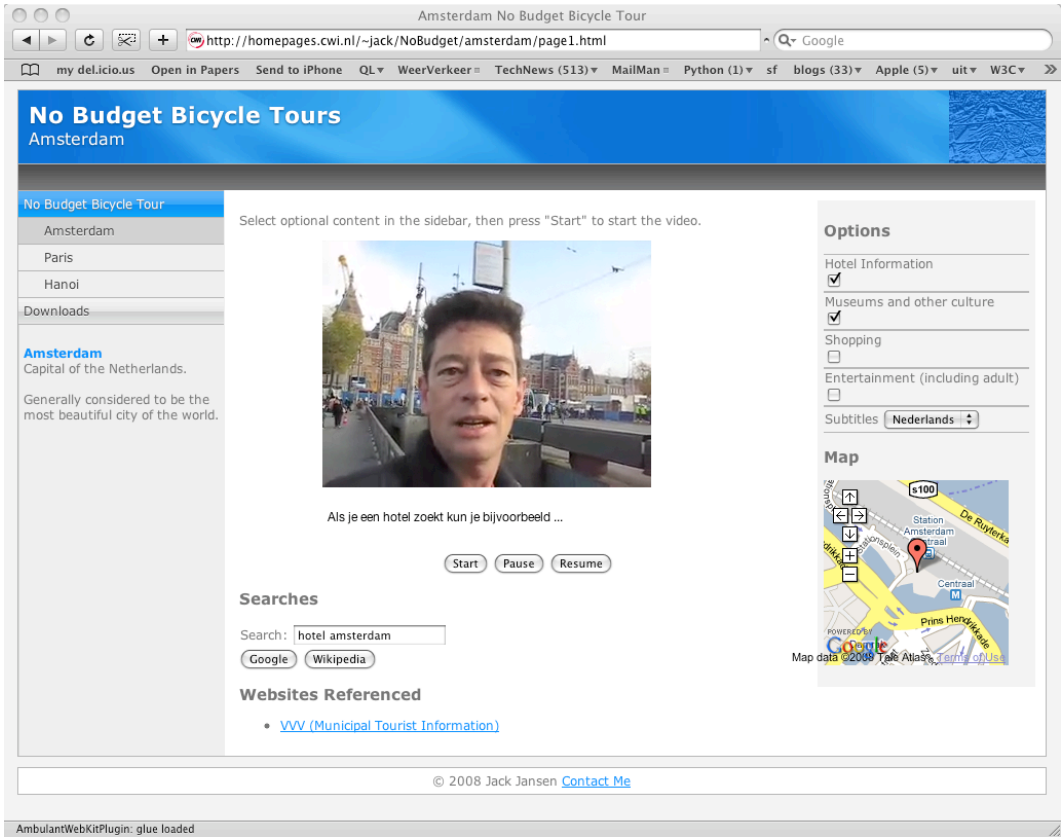
we look at existing technologies for data model support, and investigate how well these match our requirements. Section 4 describes our SMIL State solution, as well as the motivations for our design. In section 5, we report on our initial implementation of SMIL State in the CWI Ambulant open source SMIL player. In section 6 we describe two example presentations and their architec-

ture. We conclude with determining how well our solution matches our requirements, and some ideas about future work.

2.2 Scenario

In this paper we will concentrate on presentations which have time as their major structuring mechanism and that require user interaction/selection as the secondary mechanism. To set the stage, let us start with an example of the type of presentations we want to enable.

Figure 1 – Screen Shot of Guided Tour webapp



The application shown in figure 1 is a web-based guided tour through Amsterdam.[2] The backbone of the application is a video, with the tour guide showing some highlights of the city, with additional information provided from a variety of external sources on hotels, shopping opportunities, entertainment and nightlife. The application allows viewers to select the topics in which they are interested dynamically: 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. Of course, such choices are

2 A version of this example is available on the CWI Ambulant player website: http://ambulantplayer.org /smilStateExample.shtml

not static: the user should be able to change the content selection while viewing the presentation. If, in doing so, it turns out the cultural information is too detailed for his taste, he also 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 a local can) and gives only terse information on the various subjects he encounters along the way. However, for each item he describes, 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; 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, orienting the viewing within the city. This has the benefit that the user may bookmark a place of interest, or again pause the presentation to search for related interesting places in the vicinity. The application also allows for the dynamic insertion of adwords, which bring up sponsored links relevant to the material currently presented. An interesting feature is that while sponsored links are triggered by location information, they are also temporally shifted so that their presentation is delayed until after a main content stream has completed. (This delayed scheduling can obviously be used for a host of applications beyond ad insertion.)

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.

Our application example is similar in scope to the personalized multimedia tourist guide described by Scherp and Boll in [28], 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 current viewing context. 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), a significant authoring saving could be realized. It also eases the process of 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.

2.2.1 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 following requirements are important:

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

As will been seen, our SMIL State approach meets all of these requirements.

2.3 A Review of Existing Technology

Given the requirements of the previous section, this section will examine and evaluate the facilities available in existing Web technology. We will start with languages that aim at solving the whole problem space, 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.

2.3.1 Multimedia on the Web

For interactive multimedia on the web there are currently two solutions in widespread use: Flash, and JavaScript combined with a plug-in to handle media playback (such as RealPlayer or, again, Flash). SMIL, which we will examine in greater depth in the following section, is not currently a serious contender in this market because it defines an execution model that is separated from the procedural control favored in web design.

The Flash solution is by far the most common, and used by websites like YouTube and Asterpix. All interaction is programmed explicitly in ActionScript [1], requiring specialized skills and tools. Moreover, due to the binary nature of the Flash distribution, 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 mash-ups).

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 [16], or a plug-in 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 time codes) 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 [27], which can be used to create interactive accessible slideshows.

2.3.2 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 [6], 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 an XForm form designed for a desktop web browser to be reused on a mobile browser, or even a voice browser [17]). 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 [10]. 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 for which this was not the primary design goal.

2.3.3 SMIL

SMIL, the Synchronous Multimedia Integration Language 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 because of the adaptive nature of hierarchical timing. 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 within that presentation. Or, more directly: there are no variables. In addition, with events being the only dynamic 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 [14] or monads [22, 30] to enable side-effects and input/output. Without these, their application domain would have been severely limited.

2.3.4 Other Related Work

The technologies 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 [15] 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 [9] would operate on a document during load time, or dynamically, reacting to DOM events [26]. XBL has no notion of time or control flow, so using it to create self-paced multimedia presentations would be difficult.

XConnector [23] 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, such as 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 [24] 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 [4] is similar to XConnector plus XTemplate, but more limited in scope: it allows 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 [19] (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 [29], which adds a `<value>` element that can be used to store free variables. (It also adds a template 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, as well.

2.3.5 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.

Table 1 – Technology Comparison

	F	*JS*	*S2*	*S+X*	*XBL*	*KST*	*XCXT*	*XS*	*S3*
Structured	-	-	✓	✓	n/a	✓	+/-	+/-	✓
Time based	✓	-	✓	✓	-	✓	+/-	✓	✓
Fragment support	-	-	✓	✓	n/a	✓	✓	✓	✓
Variables	✓	✓	-	-	✓	✓	✓	-	✓
Language bridging	-	✓	-	-	✓	-	unknown	-	✓
Adaptability	+/-	-	✓	✓	✓	✓	✓	✓	✓
Accessibility	-	+/-	✓	✓	n/a	✓	unknown	✓	✓
Reusability	-	-	+/-	+/-	+/-	+/-	✓	+/-	✓

F: Flash; *JS*: JavaScript plus DOM access; *S2*: SMIL 2.1 (not including SMIL State); *S+X*: SMIL combined with XForms; *XCXT*: XConnector and XTemplate; *XS*: XHTML+SMIL; *S3*: SMIL 3.0 including SMIL State.

As we can see in the table, SMIL 2.1 does fairly well on the structuring front, but falls short in practical issues like rich interactivity and integration with other components (ignoring SMIL State, for the moment). 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 provides more options, 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.

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 that our work considered:

- both solutions allow for rich data structures in the data model, but where we opted for XML for easy sharing, they felt 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 below.

2.4 Design and Architecture

The main thrust of the research leading up to this article is that the addition of variables and communication would enable SMIL to be used in a number of application areas that are currently beyond its reach. These application areas include:

- *Courseware* is an important application area for multimedia software. One of the main 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 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 the ball 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. Prior to SMIL 3.0, SMIL provided *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.

A separate, but related, issue with older SMIL releases 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.

2.4.1 SMIL State Elements

SMIL State was designed using a two-tiered approach: first we architected hooks in the SMIL language to enable inclusion of a data model and expression language; we next focused on the 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;
- an event `stateChanged(ref)` that occurs when the specified item in the data model changes.

All of these hooks are modeled after existing SMIL constructs: `expr` behaves in a manner 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 existing mechanism in which SMIL animation and DOM access are allowed to modify attribute values in a running SMIL presentation (the so-called "sandwich model"). In this model, attribute value templates are only allowed in attributes where they cannot modify the time graph of a running presentation, similar to what is defined for SMIL animation.

For the default data model and expression language we have selected XML and XPath, respectively. 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 [20], 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.

Listing 1 – Sample SMIL document with SMIL State constructs highlighted

```
<smil>
 <head>
  <state>
   <data xmlns="">
    <wantAd></wantAd>
   </data>
  </state>
 </head>
 <body>
  <seq>
   <par>
    <video src="match.mp4"/>
    <img xml:id="banner" begin="10s"
```

```
      end="15s" src="banner.png"/>
    <setvalue begin="banner.activateEvent"
      ref="wantAd" value="'commercial.mp4'"/>
   </par>
   <video expr="wantAd" src="{wantAd}"/>
  </seq>
 </body>
</smil>
```

Listing 1 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 the string `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.

2.4.2 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 time graph 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 decou-

pling 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 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.

2.4.3 Content Control

SMIL has always supported optional content, the ability to render or skip content based on environmental conditions. *System tests* allow the presentation author to trigger on predefined conditions, such as available bandwidth and screen size, and *custom tests* allow extension of these with author-defined binary conditions. These constructs suffer from a number of drawbacks, however:

- there is no way to set the value of a custom test attribute from the presentation, and the user interface for defining these values is unspecified in the SMIL standard and left to individual implementations;
- the standard specifically allows system and custom tests to be evaluated once, at document load time, which limits their usefulness for interactivity and dynamic adaptation;
- the lack of an expression language means presentation authors can only test for attributes being true, not for them being false, and not for combinations of attribute values.

SMIL State integrates system tests and custom tests into its general expression framework. For example, the SMIL 2.1 construct

```
<audio src="background.ogg" systemBitrate="128000"/>
```

plays an audio fragment only if enough bandwidth is available. The corresponding SMIL State construct

```
<audio src="background.ogg" expr="smil-bitrate() &gt;
128000"/>
```

ensures dynamic evaluation, which means the presentation can adapt to varying bandwidth conditions (for example in mobile situations). Another example of the advantage of rich expressions is the ability to specify

```
<audio src="background.ogg" expr="smil-bitrate() &gt;
128000 and not(smil-audioDesc())"/>
```

This plays the background music track only if enough bandwidth is available, and if it does not interfere with audio descriptions.

2.5 Implementation

We have implemented SMIL State in our open source Ambulant SMIL player, this implementation was used to experiment with our sample applications. The implementation follows the two-tiered approach of the SMIL state design:

- architectural hooks into the SMIL language have been implemented in the core SMIL engine;
- data model and expression language are implemented in optional plug-in modules.

In this section we will look at three example expression language implementations: the *XPath-WebKit-state* module which was used for the guided tour, the *XPath-standalone-state* module used for the delayed advertisement presentation and a *Python-state* module.

Listing 2 – State component API

```
class state_component_factory {
  state_component *new_state_component(const char *uri);
};

class state_component {
  void declare_state(const lib::node *state);

  void set_value(const char *var, const char *expr);
  bool bool_expression(const char *expr);

  void want_state_change(const char *ref,
       state_change_callback *cb);
    ...
};

class state_change_callback {
  void on_state_change(const char *ref);
};
```

The basics of the interface between the core interpreter and the expression language plug-in are shown in listing 2. Each plug-in provides a `state_component_factory` interface. The core iterates over these, passing the expression language specified by the document author as a parameter. A plug-in that implements this language will return its implementation, and the iterating stops.

Now the core calls `declare_state` passing the `<state>` element to initialize the data model. During runtime, methods such as `set_value` and `bool_expression` are called to implement the corresponding SMIL State elements and attributes. `StateChanged` events are implemented by the core calling `want_state_change` to signal its interest in changes to a specific data model item. The plug-in will now call `on_state_change` whenever the item is modified.

The advantage of using plug-in modules and a factory class for implementing the data model and expression language is that it enables multiple implementations. One use for multiple implementations is the selection of the SMIL State expression language: the *Python-state* factory will return its implementation only if the SMIL author has specified that Python is to be used as the expression language. Another use of the factory class is that it allows plug-ins to dynamically determine whether they are applicable: both *XPath-WebKit-state* and *XPath-standalone-state* implement XPath as the expression language, but *XPath-WebKit-state* will only return its implementation after determining that the SMIL interpreter is currently hosted in a WebKit plug-in [4].

The *XPath-standalone-state* implementation is rather mundane: 700 lines of C++ that use the DOM and XPath facilities of the Gnome libxml2 [3] to implement SMIL State for standalone documents.

The *XPath-WebKit-state* implementation is more interesting: the application requires that it interfaces with a browser DOM and Javascript implementation as well as with an XForms implementation. Programming this directly in C or C++ using the NSAPI browser plug-in API would be painful: NSAPI is rather old, and its model is low level and verbose. As an example of how verbose it is, the following JavaScript statement obtains the base URL of the currently displayed HTML page:

```
base = document.location.href;
```

The equivalent C++ code is 50 lines long. Exporting functions from C++ to JavaScript also requires similarly verbose hand-written code.

Safari on MacOSX not only supports NSAPI-based plug-ins but also native *WebKit plug-ins*. These plug-ins are written in Objective-C, and tie in well with the AppKit and Foundation toolkits that are commonly used on OSX to create applications. Because Objective-C is a modern high-level language, it supports fairly rich introspection features, and the WebKit plug-in API exploits this to transparently bridge Objective-C to JavaScript and vice-versa: the plug-in can access JavaScript objects (and, hence, DOM objects) relatively easily, and exporting objects from Objective-C to JavaScript is similarly easy. Objective-C, in turn, is transparently bridged to Python through *PyObjC*, which is a standard component of MacOSX. And as the full Ambulant API is also transparently bridged to Python, through a modified version of the stan-

dard (but little known) Python tool *bgen,* the *XPath-WebKit-state* implementation is now a mere 200 lines of Python.

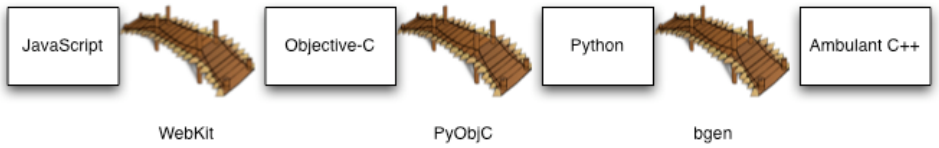


Figure 2 – Implementation language bridging

Figure 2 shows the cascade of bridges mentioned in the previous paragraph, and despite their rickety appearance we have not experienced stability issues. The use of dynamic languages and the availability of the language bridges has enabled us to use rapid prototyping methods to perform these experiments. 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 plug-in. The relevant components in this implementation are shown in figure 3.

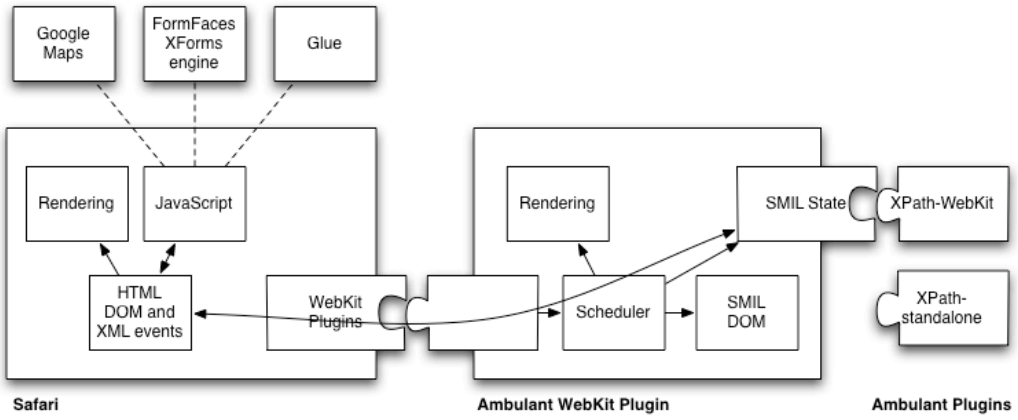


Figure 3 – Browser plug-in implementation

We have also started thinking about implementing browser integration through the standard NSAPI plug-in API, to facilitate using SMIL State in Firefox or Internet Explorer. Experience with the WebKit plug-in suggests that providing a general bridge to a high-level language may be a good solution that allows easy experimentation, so we are looking at implementing a generalized JavaScript-Python bridge based on NSAPI.

2.6 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.

2.6.1 Guided Tour Webapp

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

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 lattitude 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.

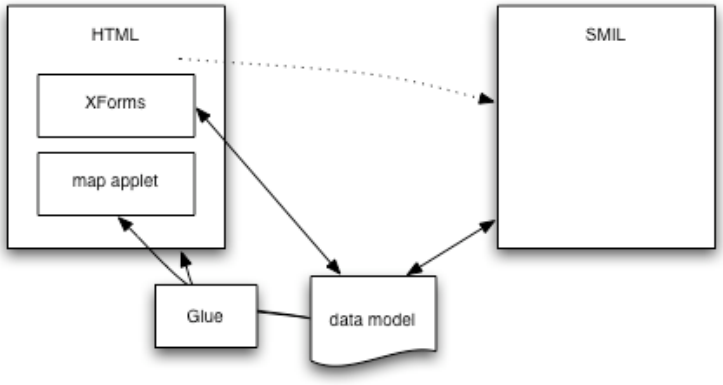


Figure 4 – Guided tour document model

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 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.

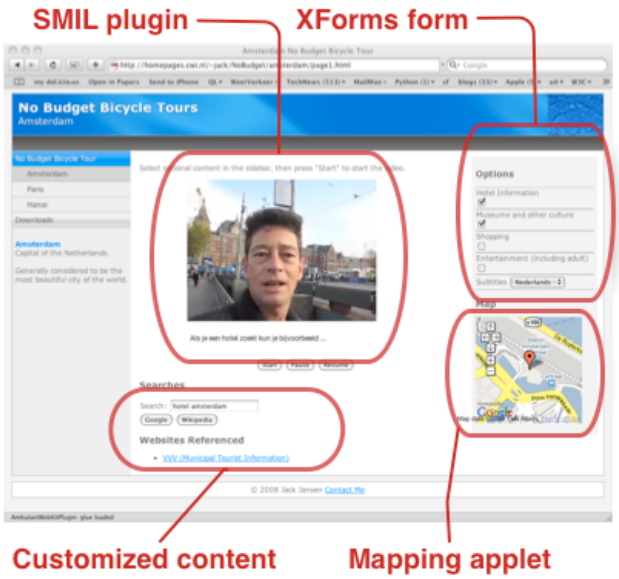


Figure 5 – Mapping of components to screen rendition

The 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 plug-in.

Listings 3 and 4 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.

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.

Listing 3 – HTML and XForms code

```
<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>
```

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.

Listing 4 – SMIL code

```
<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>
```

2.6.2 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 [11], 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 an unlikely general model.

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 target content, are generally frowned upon, and recently most major browsers contain features that actively try to forestall pop-ups and other dis-

ruptive advertisements. Instead of the forced consumption of ads, web pages tend to work with the voluntary model: the user has the option of clicking a banner ad. While even this may go too far for some people, the model probably will have a much larger acceptance than forced ads.

We feel that it would be good to transport the voluntary banner ad method to the realm of multimedia. However, if the user is in the mindset of watching a video it may be unlikely that this user clicks the advertisement instead.

To address this issue, 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.

Figure 6 – Video with banner for delayed advertisement showing in bottom right corner

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 6 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 7 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 Ford banners, and was spared the Lux ad. (At least, during the first commercial break!)

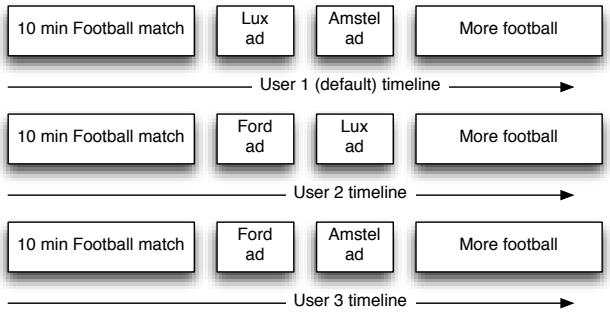


Figure 7 – Different playout orders

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 web pages. 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 listing 1 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.

2.7 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, so we only need to look at the three requirements where SMIL 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. Even so:

XPath may be a rich language to encode expressions, it is not a very user-friendly one. We plan to investigate whether it is possible to come up with an alternative for XPath that is as easy to use as KST or JavaScript while still allowing the use of a full XML document as the underlying data store.

- *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 became a W3C Recommendation in December 2008.

The model is fairly easy to support, as is demonstrated by our multiple implementations, as well as by an independent third-party implementation which was required for inclusion in the SMIL 3.0 Recommendation .

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 some of the shortcomings sketched in this section: Current web application toolkits are (naturally) oriented towards procedural languages, specifically JavaScript. A more declarative interface, possibly based on using XBL to connect widget-like components would not only benefit our model, but also help accessibility and general reuse.

We also plan to experiment with more rich interaction with the environment, through the *Python-state* implementation. This implementation should allow things like controlling and interrogating external applications, which could be put to good use for "hands-on" style courseware and such.

Distributed shared state is another area that has our interest: with the ubiquitous availability of handheld devices that have decent connectivity, compute power and rendering capabilities it is interesting to look at the possibility to create presentations that are shared among devices in a loosely coupled manner.

2.8 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.

2.9 References

1. http://www.adobe.com/devnet/actionscript/
2. http://www.macromedia.com/software/flash/about/
3. http://xmlsoft.org/
4. Apple Inc. (2008) WebKit Plug-In Programming Topics. Available at: http://developer.apple.com/documentation/InternetWeb/Conceptual/WebKit_PluginProgTopic/WebKit_PluginProgTopic.pdf .
5. Bos, B., Lie, H., Lilley, C. and Jacobs, I. (1998) Cascading Style Sheets, level 2. Available at: http://www.w3.org/TR/CSS2/.
6. Boyer, J. (2007) XForms 1.0 (Third Edition). W3C. Available on: http://www.w3.org/TR/xforms/ .
7. Bulterman, D. and Rutledge, L. (2008) SMIL 3.0: Interactive Multimedia for the Web, Mobile Devices and Daisy Talking Books. Springer-Verlag, Heidelberg, Germany, ISBN: 978-3-540-78546-0.
8. Bulterman, D. et al. (2008) Synchronized Multimedia Integration Language (SMIL 3.0). W3C. Available on: http://www.w3.org/TR/SMIL/ .
9. Clark, J. (1999) XSL Transformations (XSLT) Version 1.0. Available at: http://www.w3.org/TR/xslt.
10. Clark, J. and DeRose, S. (1999) XML Path Language (XPath) Version 1.0. Available at: http://www.w3.org/TR/xpath.
11. 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, pp. 165-172. Doi:10.1145/1166160.1166202 .
12. Ferraiolo, J., Fujisawa, J., Jackson, D. et al. (2003) Scalable Vector Graphics (SVG) 1.1 Specification. Available at: http://www.w3.org/TR/SVG11/.
13. Flanagan, D. (2006) JavaScript: The Definitive Guide. O'Reilly & Associates, Sebastopol, CA, USA. ISBN 0-596-10199-6.
14. Gifford, D. and Lucassen, J. (1986) Integrating functional and imperative programming. ACM conference on LISP and functional programming. Doi:10.1145/319838.319848 .
15. Hickson, I. (2007) XML Binding Language (XBL) 2.0. W3C. Available on: http://www.w3.org/TR/xbl/ .

16. Hickson, I. et al. (2009) HTML 5 Draft Recommendation. Available at: http://www.whatwg.org/specs/web-apps/current-work/. Retrieved on February 2, 2009.
17. 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, pp. 201-208. Doi:10.1145/1145581.1145624 .
18. Jansen, J. and Bulterman, D. (2008) Enabling adaptive time-based web applications with SMIL state. DocEng '08: Proceedings of the 2008 ACM symposium on Document Engineering (2008). ACM, New York, NY, USA. Doi: 10.1145/1410140.1410146 .
19. 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:10.1145/1030397.1030411 .
20. Lewis, R. et al. (2007) Content Selection for Device Independence (DISelect) 1.0. W3C. Available on: http://www.w3.org/TR/cselection/ .
21. 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, pp. 95-101. Doi:10.1145/317665.317681 .
22. Moggi, E. (1988) Computational Lambda-calculus and monads. In proceedings 4th Annual Symposium on Logic in Computer Science. IEEE Computer Society Press, Washington, DC.
23. Muchaluat-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:10.1145/585058.585069 .
24. Muchaluat-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:10.1080/13614560208914739 .
25. Pemberton, S. et al. (2002) XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). Available at: http://www.w3.org/TR/xhtml1.
26. Pixley, T. (2000) Document Object Model (DOM) Level 2 Events Specification Version 1.0. Available at: http://www.w3.org/TR/DOM-Level-2-Events/.
27. Raggett, D. (2006) Slidy - a web based alternative to Microsoft PowerPoint. XTech (Amsterdam, May 16-19 2006). Available on: http://www.w3.org/2006/05/Slidy-XTech/slidy-xtech06-dsr.pdf .

28. 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:10.1145/1027527.1027566 .
29. 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:10.1145/1284420.1284442 .
30. Wadler, P. (1990) Comprehending Monads. In Proceedings of the 1990 ACM Conference on Lisp and Functional Programming, pages 61--77, Nice, France, 1990.

Appendix A - SMIL 3.0 State

Editors for SMIL 3.0

Jack Jansen, CWI

Julien Quint, DAISY Consortium

This appendix contains chapter 13 of the SMIL 3.0 Recommendation. The full text of the recommendation is available at http://www.w3.org/TR/SMIL3/ .

The main body of this thesis presents a somewhat abstracted view of SMIL State, for reasons of readability. This appendix has the full, detailed specification of the functionality.

1 Overview and Summary of Changes for SMIL 3.0

This section is informative.

The modules defined in this chapter are all new modules which were not part of the SMIL 2.1 specification.

2 Introduction

This section is informative.

A SMIL 2.1 presentation has a lot of state that influences how the presentation runs. Or, to rephrase that in a procedural way, state that influences decisions that the SMIL scheduler makes. All this state is either implicit in the presentation (what nodes are active and how long into their duration they are, how many iterations of a repeat we have done, which nodes in an excl are paused because a higher priority node has preempted them, etc.), or completely external to the presentation (system tests and custom tests).

This has the effect that the only control flow that the SMIL author has at his/her disposal is that which is built in to the language, unless the SMIL engine includes some scripting language component and a DOM interface to the document that the author may use to create more complex logic.

The modules in this section provide a mechanism whereby the document author may create more complex control flow than what SMIL provides through the timing and content control modules, without having to go all the way of using a scripting language. One way to provide this is to allow a document to have some explicit state (think: variables) along with ways to modify, use and save this state.

In addition, the mechanisms that the SMIL BasicContentControl and CustomTestAttributes modules provide for testing values are limited: basically one can only test for predefined conditions being true (not for them being

false) and there is a limited form of testing for multiple conditions with "and" being the only boolean operator.

Application areas include things like quizzes, interactive adaptation of presentations to user preferences, computer-aided instruction and distant learning.

3 Relation To Other Standards

This section is informative.

The design of these modules was done after meeting with the W3C Backplane Group (part of the Hypertext Coordination Group) and various choices were influenced by the direction that group is taking.

These modules therefore borrow heavily from work done by other W3C working groups:

- The data model is a small XML document addressed with XPath 1.0 [XPATH10]. This follows the lead set by XForms 1.0 [XFORMS10].
- The XPath function interface to system and custom test data and the expr attribute are modelled after work by the Device Independent WG on DISelect 1.0 [DISELECT10].
- Attribute Value Templates used in StateInterpolation are lifted from XSLT 1.0 [XSLT10], [XSLT20].

The intention of these modules it to provide authors with the minimum functionality required to create compelling presentations, not to import all functionality from the standards they were lifted from, and concentrate on the timing integration issues. Applications requiring a richer set of primitives should import, for example, the XForms data model through the XML namespace mechanism.

4 Module Overview

This section is normative.

This chapter defines the following modules:

- StateTest, containing extended content selection;
- UserState, containing author-defined state;
- StateSubmission, saving author-defined state;
- StateInterpolation, allowing runtime modification to attribute values.

5 The SMIL StateTest Module

5.1 Changes for SMIL 3.0

This section is informative.

The SMIL 3.0 StateTest Module defined in this document is a new module which was not part of the SMIL 2.1 specification.

5.2 Overview

This section is informative.

The mechanisms that the BasicContentControl and CustomTestAttributes modules provide for testing values are limited: basically one can only test for predefined conditions being true (not for them being false) and by specifying multiple system test attributes an author has a way to simulate an *and* operator but that is all.

This module introduces a generalized expr attribute that contains an expression. If the expression evaluates to `false` the element carrying the attribute is ignored. If the expression evaluates to `true` or if there is any error (this ranges from expression syntax errors and type errors to unavailability of the expression language engine) the element is treated normally.

5.3 Elements and Attributes

This section is normative.

The expr Attribute

`expr`

This attribute contains an expression that is evaluated at runtime, i.e. each time the element becomes active. If the expression evaluates to `false` the element is ignored (including its hierarchy of descendants). If it is impossible to resolve the expression specified in the expr attribute to a boolean, user-agents must ignore the expr attribute.

Any profile using this module needs to define the language used to specify the expression.

This section is informative.

The SMIL 3.0 Language Profile specifies that XPath 1.0 is used as the default expression language, and the context in which the expressions are evaluated is as follows:

- The *context node*, *context position* and *context size* are not relevant to the functionality in the State Test module;
- The *variable bindings* are empty;
- The *function library* consists of the functions listed below (in addition to the builtin XPath functions); and
- The *set of namespace declarations* is not relevant to the functionality in the State Test module.

Alternative expression languages that could be used are a scaled down form of XPath as used by DI, or EcmaScript, Python, Lua or any other language suitable for the application domain of the profile.

Note that there is a slight but important semantic difference between using content control attributes and using expr: the latter is guaranteed to be dynamically evaluated at runtime and may therefore be used for more dynamic control whereas there is no such guarantee for the former.

5.4 Functions

This section is normative.

This module defines a set of functions for use in the expr attribute (possibly in addition to functions already defined in the expression language). The naming convention used for the functions is compatible with XPath 1.0 expressions, a profile using this module with another expression language must specify a transformation that needs to be applied to these function names to make them compatible with the expression language specified.

```
boolean smil-audioDesc()
```

Corresponds to systemAudioDesc.

```
number smil-bitrate()
```

Corresponds to systemBitrate.

```
boolean smil-captions()
```

Corresponds to systemCaptions.

```
boolean smil-component(string uri)
```

Corresponds to systemComponent, checks for availability of a single playback component.

```
boolean smil-customTest(string name)
```

Corresponds to customTest, checks the current state of the given custom test.

```
string smil-CPU()
```

Related to systemCPU. This function returns the CPU on which the user agent runs.

```
number smil-language(string lang)
```

Related to systemLanguage. The string argument should be a BCP47 [BCP47] language tag. If the language does not match any language range in the users' preferences the function returns **0**. If the tag does match the function returns a positive number, such that languages that match higher in the language priority list return a higher number.

`string smil-operatingSystem()`

Related to systemOperatingSystem. This function returns the operating system on which the user agent runs.

`string smil-overdubOrSubtitle()`

Values: `overdub` or `subtitle`

Corresponds to systemOverdubOrSubtitle.

`boolean smil-required(string uri)`

Corresponds to systemRequired.

`number smil-screenDepth()`

Corresponds to systemScreenDepth.

`number smil-screenHeight()`

Related to systemScreenSize, returns the height of the screen in pixels.

`number smil-screenWidth()`

Related to systemScreenSize, returns the width of the screen in pixels.

5.5 Examples

This section is informative.

Here is a SMIL 3.0 Language Profile example of an audio element that is only played if audio descriptions are off and the internet connection is faster than 1Mbps. Think of using it for playing background music only when this will not interfere too much with the real presentation:

```
<audio src="background.mp3"
    expr="not(smil-audioDesc()) and smil-bitrate() &gt; 1000000" />
```

Here is an example that will show the image colour.jpg to most english-speaking people. However, people preferring American English over other variants of english will see color.jpg. Non-english speaking people will see couleur.jpg.

```
<switch>
  <img src="color.jpg"
    expr="smil-systemLanguage('en-us') &gt;= smil-systemLanguage('en')"
/>
  <img src="colour.jpg" expr="smil-systemLanguage('en')" />
  <img src="couleur.jpg" />
</switch>
```

6 The SMIL UserState Module

6.1 Changes for SMIL 3.0

This section is informative.

The SMIL 3.0 UserState Module defined in this document is a new module which was not part of the SMIL 2.1 specification.

6.2 Overview

This section is informative.

This section introduces a data model that document authors may refer to in the context of the expr attribute, allowing elements to be rendered depending on author-defined values. A mechanism to change values in the data model is also included.

The actual choice of the expression language is made in the language profile. The SMIL 3.0 Language Profile requires support for the XPath 1.0 expression language (but allows use of other languages as well).

6.3 Elements and Attributes

This section is normative.

The UserState module defines the elements state, setvalue, newvalue and delvalue and the attributes language, ref, where, name and value.

The state Element

The state element sets global, document-wide, information for the other elements and attributes in this module. It selects the expression language to use and it may also be used to initialize the data model.

Initialization of the data model may be done in-line, through the contents of the state element, or from an external source through the src attribute (defined in the Media Object Modules section).

The src takes precedence over the inline content, which is only used if the src attribute is not specified or if the document it refers to cannot be found.

Initialization of the data model (including retrieval of the data) happens at the beginning of document playback. This may include modifications to the data model to make it play well with SMIL State use. Such modifications must be defined in the profile including this module.

This section is informative.

Allowing both inline content and a src attribute allows the former to be used as a fallback mechanism.

Note that beginning of document playback may be different than document parse time: depending on the user interface of the user agent a document may be played multiple times after being parsed once.

The SMIL language profile specifies that, in the case of using the XPath data model, an empty data model will be modified so that it consists of a single empty **<data/>** root element.

Element Attributes

The state element accepts the language and src attributes.

The setvalue Element

The setvalue element modifies the value of an item in the data model, similar to the corresponding element from XForms, but it takes its time behaviour from the SMIL ref element.

Note that setvalue only modifies existing items, it is therefore an error to specify a non-existing item, depending on the expression language semantics. In case of such an error SMIL Timing semantics of the element are not affected.

The setvalue supports all timing attributes, and participates normally in timing computations. The effect of setvalue happens at the beginning of its simple duration.

Element Attributes

The setvalue element accepts the ref and value attributes. Both of these are required attributes.

The newvalue Element

The newvalue element introduces a new, named, item into the data model.

The newvalue supports all timing attributes, and participates normally in timing computations. The effect of newvalue happens at the beginning of its simple duration. Depending on the semantics of the expression language it may be an error to execute the newvalue element more than once. In case of such an error SMIL Timing semantics of the element are not affected.

The ref and where determine where in the data model the new item is introduced. If the expression language does not support a hierarchical namespace these attributes are ignored. The name attribute determines the name for the new item.

Element Attributes

The newvalue element accepts the ref, where, name and value attributes. Which of these are required depends on the expression language.

The delvalue Element

The delvalue element deletes a named item from the data model.

The delvalue supports all timing attributes, and participates normally in timing computations. The effect of delvalue happens at the beginning of its sim-

ple duration. Depending on the semantics of the expression language deletion of variables may not be supported, or it may be an error to execute the delvalue element on a non-existing item. In case of such errors SMIL Timing semantics of the element are not affected.

Element Attributes

The delvalue element accepts the ref attribute.

The language Attribute

The language attribute selects the expression language to use. Its value should be a URI defining that language. The default value for this attribute is defined in the profile.

SMIL implementations should allow expression language availability to be tested through the systemComponent attribute.

The ref Attribute

The ref attribute indicates which item in the data model will be changed. The language used to specify this, plus any additional constraints on the attribute value, depend on the expression language used.

This section is informative.

The reason that newvalue has both a ref and a name attribute is that some languages, notably XPath 1.0, do not support ref referring to a non-existing named item in the data model. Therefore, name is used to give the name for the new item and ref and where specify where it is inserted. For expression languages without a hierarchical namespace ref and where should be omitted and only name is needed.

This section is informative.

For the SMIL 3.0 Language Profile the value of the ref attribute is an XPath expression that evaluates to a *node-set*. It is an error if the node-set does not refer to exactly one node.

The where Attribute

The where attribute indicates where in the data model the new item will be inserted, if the expression language supports a hierarchical data model. The allowed values are `before`, `after` and `child`, the default.

The name Attribute

The name attribute specifies the name for the new data model item. This name must adhere to constraints set by the expression language used.

The value Attribute

The value attribute specifies the new value of the data model item referred to by the ref element. How the new value is specified in the value attribute is

defined in the profile that includes this module. This specification also states whether only simple values are allowed or also expressions, and when those expressions are evaluated.

If a statically-typed language is used as the data model language it is an error if the type of the value expression cannot be coerced to the type of the item referred to by the ref.

6.4 Examples

This section is informative.

Here is a SMIL 3.0 Language Profile example of a sequence of audio clips that remembers the last audio clip played, omitting the state declaration in the head for brevity:

```
<seq>
    <audio src="chapter1.mp3" />
    <setvalue ref="lastPlayed" value="1" />
    <audio src="chapter2.mp3" />
    <setvalue ref="lastPlayed" value="2" />
    <audio src="chapter3.mp3" />
    <setvalue ref="lastPlayed" value="3" />
</seq>
```

Here is an extension of the previous example: not only is the last clip remembered but if this sequence is played again, later during the presentation, any audio clips played previously are skipped:

```
<seq>
    <seq expr="lastPlayed &lt; 1">
        <audio src="chapter1.mp3" />
        <setvalue ref="lastPlayed" value="1" />
    </seq>
    <seq expr="lastPlayed &lt; 2">
        <audio src="chapter2.mp3" />
        <setvalue ref="lastPlayed" value="2" />
    </seq>
    <seq expr="lastPlayed &lt; 3">
        <audio src="chapter3.mp3" />
        <setvalue ref="lastPlayed" value="3" />
    </seq>
</seq>
```

6.5 Data Model

This section is informative.

As stated before, the normative choice of an expression language and data model is made in the profile that includes this module, but for ease of reading this section informatively describes the choices in the SMIL 3.0 Language Profile: XPath 1.0 operating on a simple XML document contained in the state element.

It is important to note that the data model is an XML document. This is not to be confused with the variable bindings in the expression context, another

namespace that XPath has. These variable bindings are not supported through SMIL State. Therefore references to state elements are node-set expressions, not *$name*-style variable references. This usage allows for nested variables and more complex data structures than the flat namespace of the variable bindings provides. SMIL follows the lead of XForms here.

The state element, of which at most one may occur, in the head section, should either be empty or contain a well-formed XML document.

The XPath context in which the expressions are evaluated is as follows:

- The *context node* is the root of the XML document specified with the state element;
- *context position* and *context size* refer to that same element;
- The variable bindings are empty;
- The *function library* consists of the functions defined in the StateTest module and those defined in the XPath Core Function Library; and
- The set of namespace declarations is defined by the **xmlns** attribute on the context node.

This context means that an expression of the form `count` has the same meaning as one of the form `/data/count`. Moreover, the XPath type conversion rules result in `count + 1` in meaning the exact same things as `number(/data/count) + 1`.

Data Model Examples

Here is the minimal state section that corresponds to the audio clip example above:

```
<smil>
 <head>
  <state>
     <data xmlns="">
      <lastPlayed>0</lastPlayed>
     </data>
  </state>
  ...
```

Expression Constraints

The UserState module does not constrain the data model and expressions of the underlying language, unless specifically done so in a profile. For ease of reading most examples in this document use simple variable-style names, but richer constructs, such as setting attribute values with XPath or using compound values in Python, are allowed.

6.6 Data Model Events

This section is informative.

Supported events for event-based timing are normatively specified in the profile. For ease of reading we include the relevant event defined in the SMIL 3.0 Language Profile here as well. The purpose of these events is to allow document authors to create documents (or sections of documents) that restart and re-evaluate conditional expressions whenever the values underlying the expressions have changed.

`stateChange(ref)`

Raised by the state element. The parameter is a reference to an item (or multiple items) in the data model. Whenever any of the data model items referenced by the parameter is changed this event is raised. The event does not bubble.

`contentControlChange(attrname)`

Raised by the root of the SMIL document when the named Content Control test values has changed. The list of allowed values for attrname is taken from the Content Control Module attribute names. The event does not bubble.

`contentControlChange`

Raised by the root of the SMIL document when any Content Control test value has changed. The event does not bubble.

Raising the *stateChange* event on the state element instead of on the data model element itself allows for external data models (which have a distinct xmlid-space) and on non-XML data models (depending on the expression language).

If any of the Content Control test values changes both the specific event and the general event are raised. This is because for some documents the author will want to react to a change in a specific parameter (bandwidth, screensize) only, whereas for other use cases the author may want to reconfigure the whole presentation on any change.

7 The SMIL StateSubmission Module

7.1 Changes for SMIL 3.0

This section is informative.

The SMIL 3.0 StateSubmission Module defined in this document is a new module which was not part of the SMIL 2.1 specification.

7.2 Overview

This section is informative.

This section introduces a method to save author defined state or to transmit it to an external server.

7.3 Elements and Attributes

This section is normative.

The StateSubmission module defines two elements, submission and send, and the attributes submission, action, method, replace and target.

The submission Element

The submission element carries the information needed to determine which parts of the data model should be sent, where it should be sent and what to do with any data returned. The ref attribute selects the portion of the data model to transmit and in case of XPath should be a node-set expression. The default is to transmit the whole data model (in case of xpath: "/"). The other attributes are explained below.

Element Attributes

The submission element accepts the ref, action, method, replace and target attributes. The action and method attributes are required.

Depending on the method this element describes either transmission of data, reception of data or both. The ref element is ignored if no transmission happens. The replace and target attributes are ignored if no reception happens.

This section is informative

This element was lifted straight from XForms, with the accompanying attributes. Support for asynchronous submission and the corresponding events are not needed because of SMIL's inherent parallelism.

The send Element

The send element causes the data model, or some part of the data model, to be submitted to server, saved to disk or transmitted externally through some other method. It does not specify any of this directly but contains only a reference to such submission instructions.

The send supports all timing attributes, and participates normally in timing computations. The effect of send happens at the beginning of its simple duration.

Element Attributes

The send element accepts the submission attribute.

The submission Attribute

The submission attribute is an IDREF that should refer to a submission element.

The action Attribute

A URL specifying where to transmit or save the nodeset. Which URLs are allowable must take security and privacy considerations into account.

The method Attribute

How to serialize and transmit the data. Allowed values are at least `put` and `get` but may be extended by the profile.

`put` and `get` must be symmetrical, and if there is a canonical external representation for the data model language **`put`** must create that representation.

The replace Attribute

What to replace with the reply. Allowed values are `all` for the whole SMIL presentation, `instance` for the instance data, `none` for nothing.

The target Attribute

If the value of replace is `instance`, the optional target attribute specifies which part of the data model to replace. The default is to replace the whole instance.

This section is informative.

The SMIL 3.0 Language Profile includes the StateSubmission module, and it defines that the submission element must occur in the head section.

7.4 Examples

This section is informative.

Here is an example of asynchronous submission: whenever the lastPlayed item changes because another clip has been played this fact is communicated to some server.

```
<smil>
 <head>
  <state xml:id="stateid">
     <data xmlns="">
      <lastPlayed>0</lastPlayed>
     </data>
  </state>
  <submission xml:id="subid" action="http://www.example.com/savexmldoc"
              method="put" />
 </head>
 <body>
  <par>
   <send submission="subid" begin="stateid.stateChange(lastPlayed)"
         restart="always" />
   ...
   <seq end="... some interactive end condition ..." >
    <seq expr="lastPlayed &lt; 1">
     <audio src="chapter1.mp3" />
     <setvalue ref="lastPlayed" value="1" />
    </seq>
    <seq expr="lastPlayed &lt; 2">
```

```
    <audio src="chapter2.mp3" />
    <setvalue ref="lastPlayed" value="2" />
   </seq>
   <seq expr="lastPlayed &lt; 3">
    <audio src="chapter3.mp3" />
    <setvalue ref="lastPlayed" value="3" />
   </seq>
  </seq>
 </par>
```

In another presentation we could pick this value up again synchronously and use it.

```
<smil>
 <head>
  <state>
  </state>
  <submission xml:id="subid" action="http://www.example.com/loadxmldoc"
              replace="instance" method="get" />
 </head>
 <body>
  <par>
   ...
   <seq >
    <send submission="subid" />
    <seq expr="lastPlayed &lt; 1">
     <audio src="chapter1.mp3" />
     <setvalue ref="lastPlayed" value="1" />
    </seq>
    <seq expr="lastPlayed &lt; 2">
     <audio src="chapter2.mp3" />
     <setvalue ref="lastPlayed" value="2" />
    </seq>
    <seq expr="lastPlayed &lt; 3">
     <audio src="chapter3.mp3" />
     <setvalue ref="lastPlayed" value="3" />
    </seq>
   </seq>
  </par>
```

That last example is actually a procedural roundabout way to get the same effect as using `<state src="http://www.example.com/loadxmldoc" />` without submissions.

8 The SMIL StateInterpolation Module

8.1 Changes for SMIL 3.0

This section is informative.

The SMIL 3.0 StateInterpolation Module defined in this document is a new module which was not part of the SMIL 2.1 specification.

8.2 Overview

This section is normative.

This section introduces a mechanism whereby document authors may use values from the data model to construct attribute values at runtime. The mechanism has been borrowed from XSLT attribute value templates.

Substitution is triggered by using the construct `{expression}` anywhere inside an attribute value. The expression is evaluated, converted to a string value and substituted into the attribute value.

This substitution happens when the element containing the attribute with the `{expression}` attribute becomes active.

If any error occurs during the evaluation of the expression no substitution takes place, and the `{expression}` construct appears verbatim in the attribute value.

If a profile includes this module it must list all attributes for which this substitution is allowed. It must use the same expression language for interpolation as the one used for StateTest expressions.

8.3 Elements and Attributes

This section is normative.

This module does not define any new elements or attributes.

This section is informative

The SMIL 3.0 Language Profile includes the StateInterpolation module. It allows its use in the same set of attributes for which SMIL animation is allowed plus the src, href, clipBegin and clipEnd attributes. It disallows its use on the Timing and Synchronization attributes. Its use on other attributes is implementation-dependent.

8.4 Examples

This section is informative.

This SMIL 3.0 Language Profile example shows an icon corresponding to the current CPU on which the user views the presentation, or a default icon for an unknown CPU:

```
<switch>
  <img src="cpu-icons/{smil-CPU()}.png" />
  <img src="cpu-icons/unknown.png" />
</switch>
```

8.5 StateInterpolation, Animation and DOM access

This section is normative.

Because StateInterpolation may also change attribute values its interaction with animation and DOM access needs to be defined, the so-called "sandwich model". StateInterpolation sits between DOM access and animation, i.e. DOM access will see the `{expression}` strings verbatim and it may set these values too. SMIL animation will operate on the value of the expression.

Appendix B - SMIL State Test suite

This appendix contains the interoperability tests used during the SMIL 3.0 standardization process. Two independent implementations of the SMIL State specification (one of them ours) have passed this test suite. The intention of the test suite is only to confirm that the language in the specification is such that at least two independent parties arrive at compatible implementations.

The full test suite is available at http://www.w3.org/2007/SMIL30/testsuite/ .

The appendix is included as a clarification of the interoperability procedure referred to in section 1.7.

test 01 - Attribute Value Templates

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
Chapter: SMIL 3.0 State
Module: StateInterpolation
Feature: Use of attribute value templates
File Name: test-01-avt.smil
Description: An AVT construct is used to interpolate text into a data
url.
Expected Behavior: It should display "the number 42 should be forty-two".

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
    baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
     <data xmlns="">
      <foo>42</foo>
     </data>
    </state>
  </head>
  <body>
   <seq>
    <text dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-two" />
   </seq>
  </body>
</smil>
```

test 02 - setvalue element

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
```

```
Chapter: SMIL 3.0 State
Module: UserState
Feature: setvalue element
File Name: test-02-setvalue.smil
Description: Computes expressions and sets valriables with setvalue.
       StateInterpolation is required for this test.
Expected Behavior: The output is self-describing, displayed in 3 5-second
texts.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
    baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
     <data xmlns="">
      <foo>42</foo>
     </data>
    </state>
  </head>
  <body>
   <seq>
    <text dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-two" />
    <setvalue ref="foo" value="43" />
    <text dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-three" />
    <setvalue ref="foo" value="foo+1" />
    <text dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-four" />
   </seq>
  </body>
</smil>
```

test 03 - newvalue element

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
Chapter: SMIL 3.0 State
Module: UserState
Feature: newvalue element
File Name: test-03-newvalue.smil
Description: Computes expressions and sets variables with newvalue.
       StateInterpolation is required for this test.
Expected Behavior: The output is self-describing, displayed in 3 5-second
texts.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
    baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
      <region xml:id="text" width="400" height="100" />
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
```

```
     <data xmlns="">
     </data>
    </state>
  </head>
  <body>
   <seq>
    <newvalue ref="/data" name="foo" value="42" />
    <text region="text" dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-two" />
    <setvalue ref="foo" value="43" />
    <text region="text" dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-three" />
    <setvalue ref="foo" value="foo+1" />
    <text region="text" dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-four" />
   </seq>
  </body>
</smil>
```

test 04 - state element with src attribute

```
<?xml version="1.1"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 3.0 Language//EN"
"http://www.w3.org/2007/07/SMIL30/SMIL30Language.dtd">
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
Chapter: SMIL 3.0 State
Module: UserState
Feature: state element with src attribute
File Name: test-04-statesrc.smil
Description: Obtains state variables from external document.
       StateInterpolation is required for this test.
Expected Behavior: The output is self-describing.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0" baseProfile="La-
nguage">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116"
src="test-04-statesrc-state.xml" />
  </head>
  <body>
   <seq>
    <text dur="5s"
src="data:,the%20number%20{foo}%20should%20be%20forty-two" />
   </seq>
  </body>
</smil>
```

test 05 - send element

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
```

```
Chapter: SMIL 3.0 State
Module: StateSubmission
Feature: submission and send elements
File Name: test-05-send.smil
Description: Saves the state document to a file.
Expected Behavior: After running the test there should be a file
       "test-05-send-out.xml". The contents of this file should be
       identical to "test-05-send-out-correct.xml" (modulo whitespace).

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0" baseProfile="La-
nguage">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
     <data xmlns="">
      <foo>42</foo>
     </data>
    </state>
    <submission xml:id="subid" method="put"
      action="test-05-send-out.xml" />
  </head>
  <body>
   <seq>
    <text dur="5s"
      src="data:,writing%20xml%20to%20test-05-send-out.xml" />
    <send submission="subid" />
    <text dur="5s"
      src="data:,write%20to%20to%20test-05-send-out.xml%20done" />
   </seq>
  </body>
</smil>
```

test 06 - newvalue element

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
Chapter: SMIL 3.0 State
Module: UserState
Feature: newvalue where attribute
File Name: test-06-newvalue.smil
Description: Saves the state document to a file.
       StateSubmission is required for this test to work.
Expected Behavior: After running the test there should be a file
       "test-06-newvalue-out.xml". The contents of this file should be
       identical to "test-06-newvalue-out-correct.xml" (modulo
whitespace).

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
    baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
```

```
     <data xmlns="">
      <fortytwo>42</fortytwo>
     </data>
    </state>
    <submission xml:id="subid" method="put"
      action="test-06-newvalue-out.xml" />
  </head>
  <body>
   <seq>
    <newvalue ref="fortytwo" where="before" name="fortyone" value="41" />
    <newvalue ref="fortytwo" where="after" name="fortythree"
      value="43" />
    <newvalue ref="fortytwo" where="child" name="fortytwopointfive"
      value="42.5" />
    <text dur="5s"
      src="data:,writing%20xml%20to%20test-06-newvalue-out.xml" />
    <send submission="subid" />
    <text dur="5s"
      src="data:,write%20to%20to%20test-06-newvalue-out.xml%20done" />
   </seq>
  </body>
</smil>
```

test 07 - expr attribute

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
Chapter: SMIL 3.0 State
Module: StateTest
Feature: expr attribute
File Name: test-07-expr.smil
Description: Tests expressions with expr.
       Requires UserState.
Expected Behavior: You should see two 5-second self-explanatory texts.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
    baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
     <data xmlns="">
      <foo>42</foo>
     </data>
    </state>
  </head>
  <body>
   <seq>
    <text dur="5s" expr="foo==42"
      src="data:,this%20is%20the%20first%20you%20should%20see" />
    <text dur="5s" expr="foo&lt;41"
      src="data:,THIS%20YOU%20SHOULD%20NOT%20SEE" />
    <text dur="5s" expr="foo&gt;41"
      src="data:,this%20is%20the%20second%20you%20should%20see" />
   </seq>
  </body>
```

```
</smil>
```

test 08 - expr attribute with language function

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
Chapter: SMIL 3.0 State
Module: StateTest
Feature: expr attribute
File Name: test-08-language.smil
Description: Tests for user language preference equality and inequality.
Expected Behavior: You should see one 5 second statement about english
       being one of your preferred languages (or not), another 5 second
statement
       about french. These statements should be correct.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
   baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
     <data xmlns="">
      <foo>42</foo>
     </data>
    </state>
  </head>
  <body>
   <seq>
    <text dur="5s" expr="smil-language('en')"
    src="data:,you%20should%20see%20this%20if%20you%20speak%20english" />
    <text dur="5s" expr="not(smil-language('en'))"
src="data:,you%20should%20not%20see%20this%20if%20you%20speak%20english"
/>
    <text dur="5s" expr="smil-language('fr')"
src="data:,you%20should%20see%20this%20if%20you%20speak%20french" />
    <text dur="5s" expr="not(smil-language('fr'))"
src="data:,you%20should%20not%20see%20this%20if%20you%20speak%20french"
/>
   </seq>
  </body>
</smil>
```

test 09 - state element boundary conditions

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: March 12, 2008
Chapter: SMIL 3.0 State
Module: UserState
Feature: state element with src attribute referring to nonexistent docu-
ment
File Name: test-09-statesrc-badurl.smil
```

```
Description: Test that referring to a non-existent url in the state ele-
ment uses the
    inline document as fallback.
       StateInterpolation is required for this test.
Expected Behavior: The output is self-describing.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
    baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116"
      src="test-09-statesrc-state.xml">
     <data xmlns="">
      <foo>42</foo>
     </data>
    </state>
  </head>
  <body>
   <seq>
    <text dur="5s"
      src="data:,the%20number%20{foo}%20should%20be%20forty-two" />
   </seq>
  </body>
</smil>
```

test 10 - state element boundary conditions

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: March 12, 2008
Chapter: SMIL 3.0 State
Module: UserState
Feature: state element with src attribute and inline state document
File Name: test-10-statesrc-override.smil
Description: Test that the external state document overrides the inline
one.
       StateInterpolation is required for this test.
Expected Behavior: The output is self-describing.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
   baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116"
      src="test-10-statesrc-state.xml">
     <data xmlns="">
      <foo>42</foo>
     </data>
    </state>
  </head>
  <body>
   <seq>
```

```
    <text dur="5s"
     src="data:,the%20number%20{foo}%20should%20be%20twenty-four" />
   </seq>
  </body>
</smil>
```

test 11 - events

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: January 22, 2008
Chapter: SMIL 3.0 State
Module: UserState
Feature: expr attribute
File Name: test-11-statechanged.smil
Description: Tests stateChanged event. Seeing something longer than ex-
pected
    means some expected event did not fire, seeing something shorter
means
    some event fires too often.
       Requires UserState.
Expected Behavior: You should see two 5-second self-explanatory texts.

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
   baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state xml:id="mystate"
      language="http://www.w3.org/TR/1999/REC-xpath-19991116">
     <data xmlns="">
      <foo>0</foo>
      <bar>0</bar>
     </data>
    </state>
  </head>
  <body>
   <par>
    <seq>
     <setvalue begin="5s" ref="foo" value="42"/>
     <setvalue begin="5s" ref="bar" value="43"/>
     <setvalue begin="5s" ref="foo" value="44"/>
     <setvalue begin="5s" ref="bar" value="45"/>
    </seq>
    <seq>
     <text
src="data:,you%20should%20see%20this%20from%20begin%20to%205%20seconds"
           end="mystate.stateChange(foo)" />
     <text
src="data:,you%20should%20see%20this%20from%205%20to%2010%20seconds"
           end="mystate.stateChange(//*)" />
     <text
src="data:,you%20should%20see%20this%20from%2010%20to%2020%20seconds"
           end="mystate.stateChange(bar)" />
    </seq>
   </par>
  </body>
```

```
</smil>
```

test 12 - xml document generation

```
<?xml version="1.1"?>
<!--
Copyright: Copyright 1998-2007  W3C (MIT, ERCIM, Keio), All Rights Re-
served. See http://www.w3.org/Consortium/Legal/.
Author: Jack Jansen (CWI)
Version: March 27, 2008
Chapter: SMIL 3.0 State
Module: StateSubmission
Feature: submission and send elements
File Name: test-12-empty.smil
Description: Create a variable in an initially empty state document and
save to file.
Expected Behavior: After running the test there should be a file
       "test-12-empty-out.xml". The contents of this file should be
       identical to "test-12-empty-out-correct.xml" (modulo whitespace).

-->
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0"
   baseProfile="Language">
  <head>
    <layout>
      <root-layout width="400" height="100" backgroundColor="white"/>
    </layout>
    <state language="http://www.w3.org/TR/1999/REC-xpath-19991116">
    </state>
    <submission xml:id="subid" method="put"
     action="test-12-empty-out.xml" />
  </head>
  <body>
   <seq>
    <newvalue name="foo" value="42" />
    <text dur="5s"
      src="data:,writing%20xml%20to%20test-12-empty-out.xml" />
    <send submission="subid" />
    <text dur="5s"
      src="data:,write%20to%20to%20test-12-empty-out.xml%20done" />
   </seq>
  </body>
</smil>
```

Appendix C - Acronyms

CSS	Cascading Style Sheet
DISELECT	Device Independent Selection
DOM	Document Object Model
DSL	Domain Specific Language
HTML	Hypertext Markup Language
MATHML	Mathematical Markup Language
MTAP	Multimedia Tools and Applications
RDF	Resource Description Framework
RWAB XG	Rich Web Applications Backbone Expert Group
SCXML	State Chart XML
SMIL	Synchronous Multimedia Integration Language
SVG	Structured Vector Graphics
SYMM WG	Synchronous Multimedia Working Group
URL	Universal Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations