Multimedia Systems
DOI 10.1007/s00530-011-0233-1

REGULAR PAPER

Analysis and verification of SMIL documents

Ombretta Gaggi · Annalisa Bossi

Received: 3 December 2009 / Accepted: 25 March 2011

Abstract In this paper, we consider the problem of automatic verification of SMIL documents and present a tool which can assist the user in the complex task of authoring a multimedia presentation. The tool is based on a formal semantics defining the temporal aspects of SMIL elements by means of a set of inference rules. The rules, in the spirit of Hoare's semantics, describe how the execution of a piece of code changes the state of the computation of a player. If any temporal conflict is found, the system returns a message to the user pointing out the element which contains the conflict and its motivation. This helps the user to develop robust and clear code.

Keywords SMIL · Authoring multimedia presentations · Verification · Consistency checking

1 Introduction

The authoring and design of a *multimedia presentation* is a complex and error-prone activity, especially when the temporal structure of a multimedia document increases in complexity thus increasing the possibility to introduce a temporal conflict in the synchronization constraints. Moreover, the playback of the media objects may be completely changed by the user, who can follow a link, click on an image, or even simply move the mouse over or out of an item.

Many researchers have proposed different solutions for this problem, by defining models (e.g., temporal relations [1], AHM [13]), languages (e.g., SMIL [7]) and tools (e.g., [6]). Despite these efforts, the authoring of a complex multimedia presentation is far away from being considered as easy as using a word processor or a drawing program. A step towards a solution was the introduction of the standard language SMIL for the output. SMIL greatly helps the author to develop structured and clear code. Unfortunately, it admits the presence of double inconsistent definitions inside a multimedia document. For example, consider the element `<text id=''txt01'' begin=''3s'' end=''5s'' dur=''5s''/>` which displays a text at time instant 3 and removes it after 2 s even if its defined duration is equal to 5. Another example is the sequence:

```
<seq dur="5s">
    <img id="img01" dur="5s" />
    <img id="img02" dur="5s" />
</seq>
```

which describes a sequence of two images, each one visualized for 5 s when the overall sequence, according to its attribute `dur`, lasts for only 5 s, i.e., image `img02` is not displayed at all. Possibly, this is exactly what the author expects but, if not, it is important to individuate the existence of the conflict: we think that the author would not have included `img02` in the presentation if her/his real intention is not to display it, and we are not sure that 2 s are sufficient to read the text message in the first example.

Communicated by Dick Bulterman.

O. Gaggi (✉)
Deparment of Pure and Applied Mathematics,
University of Padua, via Trieste, 63, 35121 Padua, Italy
e-mail: gaggi@math.unipd.it

A. Bossi
Department of Computer Science, University Ca' Foscari
of Venice, via Torino, 155, 30172 Mestre, Venice, Italy
e-mail: bossi@dsi.unive.it

We call a *semantic error* the presence of a *temporal conflict* in the definition of elements and attributes, like double inconsistent definition of attributes of the same object or wrong definition of the temporal structure of the document, like in the above examples. This is indeed a very common type of error, difficult to find out and to be fixed when the complexity of the presentation increases [11]. We must note here that, according to the SMIL specifications, these are not errors, and semantic rules, that govern the behavior in this case, exist. However, in a hand-authored document the presence of such a statement is due to a mistake in most cases. Therefore, a tool which points out this situation is useful. It is important to note that we do not consider the partial playback of a (group of) media item(s) as an error, but only a double, inconsistent, definition of its duration by the definition of, at least two, attributes.

Semantic mistakes are particular dangerous, because they usually cannot be automatically correct because they point out a contradiction in the definition of the behavior of media items, and require a user decision to be solved. We think that a good authoring system should also point out semantic mistakes and assist the user while fixing them.

This paper presents a tentative approach to the formulation of an axiomatic semantics for the verification of SMIL 3.0 elements. On this basis, we have developed a tool for the automatic analysis and verification of SMIL documents. We do not aim at creating a new authoring tool, but a module, which can be used both in conjunction with an authoring system, and as the basis for the development of an efficient player, since our tool allows also to find out the begin and end times of media items as discussed in Sect. 3.

Three classes of users may take advantage from the introduction of our semantics:

- *web multimedia designers*, who can check if their presentations contain mistakes or investigate the reason for unexpected behaviors;
- *developers of SMIL player*, who can use the semantics as a complement to the standard specification to check the correctness of the player with respect to the sequence of playback of media items returned by our tool. At the current state of implementation, some players do not follow the correct interpretation of the semantics of SMIL elements, therefore, in the presence of temporal conflicts, even simple multimedia documents may have different behaviors according to the chosen player. This situation does not help the authors because it is not always clear if the misbehavior is due to a semantic conflict or to a bug [11]. This means that in case of semantic mistakes, the final behavior is almost unpredictable [15, 18, 20] and the above examples show that built-in decisions, performed by some authoring systems [19], are not always a good solution;
- *developers of visual authoring system for SMIL*, who can use the correct sequence of start and end times generated by our tool to implement a preview window. Moreover, our *Semantic Validator Module* detects the presence of conflicts in a presentation and points out the wrong values. This information allows an authoring system to help the user while correcting it. We must note here that the implementation of the *Semantic Validator Module* as a separated module offers a helper to be used if the presentation does not play as expected and avoid forcing the users to change their preferred interface for authoring.

The use of the tool as basis for a player is particularly interesting since most available players are often unstable or not free of charge as reported by Eidenberg [11]. The major problem is a robust resolution of start and end times of elements, which is exactly the output generated by our tool for consistent documents. Moreover, the a priori detection of a mistake allows the player to avoid playing wrong presentations.

Our formal semantics for the SMIL language is defined by means of a set of inference rules inspired by Hoare logic [14]. The central feature of Hoare logic is the Hoare triple which describes how the execution of a piece of code changes the state of the computation. This choice carries the advantage that the SMIL structure can be enriched by assertions, expressing the temporal properties, which can be used during the authoring phase when media items are collected in more complex constructs. For example, our tool can verify the consistency of a multimedia presentation resulting from a context adaptation process. In this case, the document is dynamically built up by selecting media items compatible with many different situations in which a multimedia presentation can be played, in terms of availability of resources (e.g., network bandwidth, CPU time), device type (e.g., desktop, laptop, cell phone) and properties (e.g., screen size, number of colors). This process often generates conflicts which must be solved to guarantee the playback.

Because we compose a multimedia presentation by nesting an SMIL element into another, our rules allow us to compose the semantics by evaluating a single element inside a more complex nesting. In other words, the proposed semantics is compositional and helps the author to modularize her/his work, thus mastering the complexity of the verification of a multimedia presentation consistency.

We must note here that this paper does not aim at augmenting or correcting the SMIL specification. It aims at offering a formal semantics which can guide SMIL developers, thus improving the standard specification. An initial subset of the semantics was presented at the ACM Multimedia Conference [4].

The paper is organized as follows. In the next section, we describe the background and provide few preliminary definitions. Section 3 presents the axiomatic semantics for SMIL language which is at the basis for development of the verification tool described in Sect. 4 We conclude in Sect. 5.

2 Background and definitions

Since SMIL's first appearance, many authoring tools and players have been implemented [7] offering their users different facilities like visual editors or preview windows. Bulterman and Hardman [6] discuss the issues to be considered by authoring environments in general and conclude that one of the reason of the lower than hoped diffusion of the SMIL standard is "*... the high complexity of authoring interactive multimedia in a more abstract, transformable manner*".

In most cases, the available tools completely lack the presence of a helper to discover and find mistakes: they usually validate only the *syntactic* correctness of the document, and do not check its temporal consistency. We think that this problem is partially due to the lack of a formal semantics for the SMIL language, which can be interpreted differently from different developers. Moreover, as reported by Muriel Jourdan, one of the editors of the SMIL 2.0 Timing and Synchronization Module [17], "*... SMIL 2.0 complexity is so great that rejecting the use of formal supports gives rise to a difficult-to-read specification that cannot be free from inconsistency*" [15].

The problem of finding out temporal conflicts into SMIL documents has been already considered in literature. Sampaio et al. describe RT-LOTOS [18, 19, 20], a formal description of SMIL elements, based on automata, which enables the generation of a valid scheduling for its rendering, considering QoS problem. Unlike our approach, the authors do not define a semantics for SMIL language, but compare different players' behaviors which are still implementation dependent. This formal representation of SMIL documents can find out temporal inconsistencies, but it does not help the author to fix them. Moreover, the approach is not extensible: the automaton which describes the obtained behavior needs to be re-built after any changes.

Jourdan [15] presents the first attempt to define a formal semantics for SMIL. This approach is based on the use of timed automata and has been used during the design of SMIL 2.0 to improve specification. This paper mainly focuses on SMIL 1.0 and takes into consideration only two new features of SMIL 2.0. Unfortunately, the adopted formalism does not appear to be scalable to easily cover all the features of the third version of the standard and does not help the user to fix mistakes.

Other authors propose the use of Petri Nets to describe the temporal evolution of an SMIL document. All the approaches translate the SMIL synchronization elements into transitions and places, therefore they are required to re-built the net after each change in the presentation. The Real Time Synchronization Model (RTSM) [21] tries to detect possible temporal conflicts but it is limited to SMIL 1.0 features. The work proposed by Mazouz et al. [16] is able to find out a more complete set of inconsistencies and the presence of non-reachable media and such information is useful to correct mistakes in SMIL documents, but the system does not show the author where the error is. Moreover, the approach suffers from the explosion problem: the computation time increases exponentially with the document size. More recently, Bouyakoub et al. [2, 5] treat also some spatial elements (the `area` element) with Petri Nets. Despite the implementation of an authoring tool for correct SMIL documents, the set of behaviors that can be produced is still limited, since they do not consider significant SMIL features, like the `excl` element. Differently from other approaches, Yu et al. [22] defines a formalism based on the Petri Nets named SAM (Software Architecture Model), which does not check the presence of semantic conflicts but the satisfiability of QoS properties. The Petri Net is translated into a set of axioms which are used to prove the logical formulas which express the QoS requirements.

Summing up, as illustrated in Table 1, all the approaches described are limited to a small subset of SMIL elements, require to translate them into a specific formalism and, even if they allow to discover time conflicts, they do not always help the user to correct them. Our approach aims to overcome these limitations by defining a semantics inspired by the Hoare logic. This choice in fact allows to analyze the SMIL elements without any translation and to suggest a correction for discovered mistakes. Moreover, our semantics covers a broad set of SMIL 3 elements and attributes, and is easily scalable since, due to the compositionality of the approach, support to other elements or attributes can be obtained, in the future, by adding new rules, without reconsidering the entire system.

2.1 A definition of an abstract player

The inference rules defining the formal semantics describe how the execution of a piece of SMIL code changes the state of the playback of a multimedia presentation. The notion of *state* of a player (or of the presentation's playback) underlying our model is determined by a set of

Table 1 Comparison of the related work with our approach

	Formalism	Error detection	Help to correct	Need translation	Scheduling as output
RT-Lotos	Automata	Y (SMIL 2)	N	Y	Y
Jourdan	Automata	Y (SMIL 1++)	N	Y	N/A
RTSM	Petri Nets	Y (SMIL 1)	Y	Y	Y
Mazouz, Bouyakoub	Petri Nets	Y (SMIL 2, no `excl`)	P	Y	Y
SAM	Petri Nets	N	N	Y	N/A
Our approach	Axioms	Y (SMIL 3)	Y	N	Y

Y yes, *N* no, *P* partially, *N/A* no information available

particular values describing significant aspects of media items. Because we are interested in describing only those aspects that might influence temporal consistency, a state describes *significant time instants*: the start and end time instants of all SMIL elements contained in the presentation as well as the duration of each continuous object and the user interaction captured by the player.

Most of this information can be retrieved directly from the SMIL documents. The only useful information which is missing concerns the natural duration of each continuous media and the events due to user interactions. More precisely, by *natural duration* we mean the number of time instants for which a continuous media item plays in absence of user interaction or other temporal specifications.

A player enters into a new state in response to an event. The SMIL specification considers two types of events, *interactive* events, i.e., the user interactions like a click on an image or the movement of the mouse, and *non-interactive* events, i.e., events due to SMIL synchronization, e.g., the start or the end of a media object. The state of a player must record all this information.

Definition 1 *(State of a player) The state σ of an abstract player is a triple $\langle c, \delta, \tau \rangle$ where*

- *c is a clock, i.e., a value that records time progression*;
- *δ is the function "description" which maps an identifier id to a triple, i.e., $\delta(id) = \langle b_{id}, e_{id}, dur_{id} \rangle$ where b_{id}, e_{id}, and dur_{id} are real numbers denoting the start time, the end time and the natural duration of the element identified by id*;
- *τ is the function "event time" which records the last instant in which an event occurred.*

The state of the player does not contain the actual duration of a media item since it can be calculated as difference between its start and end time. This is not possible for the natural duration.

We note here that the function τ records only the interactive events and not the begin and end of elements and media (non-interactive events) which are fully described by the function *"description"* δ. Interactive events may involve a media item, like a click on an item or the movement of the mouse over it, but there are also user interactions which are not related to a specific item like the user keying a key in the keyboard. In the first case, the function requires as input the type of the event and the *id* of the item, in the second case, the *id* is absent. A partial list of supported events can be found in Table 2.

Table 2 List of SMIL elements, attributes and abbreviations used in this work

SMIL elements	text, img, video, audio, animation, brush, ref
	par, seq, excl
Attributes	begin, end, dur
Admitted Values	begin: a positive time value t, an event ev, ev+t, list-of-values
	end: a positive time value t, an event ev, ev+t, indefinite, list-of-values
	dur: a positive time value t, indefinite
Abbreviations	media $\overset{def}{=}$ cont | static | ref;
	cont $\overset{def}{=}$ video | audio | animation;
	static $\overset{def}{=}$ text | img | brush;
	cmd $\overset{def}{=}$ media | par | seq | excl;
	m $\overset{def}{=}$ id="m"
	ev $\overset{def}{=}$ id.begin/end, id.activateEvent/click, id.mouseover/mouseout, accesskey('c'), etc.

As described below, interactive events may occur several times, e.g., a user may click at different moments on a button. Therefore, a media item can be played several times in response to a user interaction. To represent this fact, we could have associated with each identifier a sequence of tuples representing its various executions. We choose a different solution, i.e., to create a new name each time a media item is played. This means that we can have multiple identifiers referring to different activations of the same item: they refer to the same file but are considered completely different from the synchronization point of view. The new names are generated by the function v; thus if $\mathtt{c}$ identifies a media item then $v(\mathtt{c}), v(v(\mathtt{c})), \ldots v^i(\mathtt{c}), \ldots$ are different identifiers for different executions of the same media item $\mathtt{c}$.

Table 3 List of functions and notations used in the definition of the proof rules

Function	Pre/post condition	Description
$t_{cr} : \mathbb{I} \rightarrow \mathbb{R}$	Pre	returns the current time instant in which the SMIL element *id* is evaluated
$dur : \mathbb{I} \rightarrow \mathbb{R}$	Pre	returns a real value representing the time interval for which a continuous media item plays
$times : (\mathcal{E} \times (\mathbb{I} \cup \{\perp\})) \rightarrow \mathbb{R}^*$	Pre	returns the sequence of all the time instants in which an event occurs
$time : (\mathcal{E} \times (\mathbb{I} \cup \{\perp\})) \rightarrow \mathbb{R}^\perp$	Pre	returns the time instant of the next occurrence of an event
$begin : \mathbb{I} \rightarrow \mathbb{R}$	Post	returns the time instant media item *id* starts
$end : \mathbb{I} \rightarrow \mathbb{R}$	Post	returns the time instant media item *id* ends
Abbreviation	Description	
$begin_B(\mathtt{c})$	denotes t if $\{begin(\mathtt{c}) = t\} \subseteq B$	
$end_B(\mathtt{c})$	denotes t if $\{end(\mathtt{c}) = t\} \subseteq B$	

Our framework supports all the three time container `par`, `seq` and `excl`, and attributes `begin` (negative offsets are not allowed), `end` and `dur` as reported in Table 2. Currently, our framework does not cover the whole SMIL language but, compared with the related work, it is the most complete and the only one to support the third version of the language (see Table 1). Moreover, the attributes which are missing can be easily added since the framework is built to be scalable. Since we are interested in the synchronization of multimedia presentations, we do not consider SMIL elements for layout, transitions, etc.

2.2 The assertion language

The rules provide an axiomatic semantics for the temporal aspects of SMIL elements in the spirit of Hoare logic. They allow us to derive judgements in the form of triplets:

$$\{P\}\, c\, \{Q\}$$

where P and Q are assertions, respectively, the *precondition* and the *postcondition*, and c is an SMIL element.

The assertion language used to express pre/post conditions includes a set of basic functions representing the significant temporal aspects of the media. Assertions are formed by sets of constraints on values returned by these functions. Table 3[1] lists all the functions and abbreviations used in the assertions. For instance, we write $begin(\mathtt{c}) = 10$ to mean that the media `c` starts its execution at clock time 10. Given an assertion B which contains the equality $begin(\mathtt{c}) = t$ (or $end(\mathtt{c}) = t$), we use also the notation $begin_B(\mathtt{c}) (or\, end_B(\mathtt{c}))$ to denote the time instant t occurring in the corresponding equality.[2] We note here that the SMIL language allows multiple formats of legal clock values, e.g. the values 00:02:33, 2:33 and 153 s represent the clock value 2 min and 33 s. Since they can be easily translated in the real number representing the number of seconds, we suppose that all our functions return a real value.

The assertion language contains also the functions $t_{cr}(id)$ that represents the current time instant in which the SMIL element *id* is evaluated, i.e., the clock of the state in which, considering a player executing the presentation, the player evaluates that command. The SMIL Specifications call this time the *implicit syncbase*. We use it in the precondition.

The occurrence of user interactions is represented in the precondition of a element by equalities in the form $times(event) = (t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are the next time instants in which the event will occur. The time instant in which an event occurs is the value of the clock when the player registers the occurrence of that event. We use also the functions $time(event)$ to represent just the next occurrence of the event, that is, the first element in the sequence $(t_1, \ldots, t_n)$.

Let σ be a state and A an assertion, we say that σ satisfies the assertion A, and write $\sigma \models A$, if the constraints on real values obtained by applying to A the following transformations hold:

- each occurrence of $begin(id)$, $end(id)$, $dur(id)$ and $t_{cr}(id)$ is replaced by the corresponding component of the state of the player;
- each occurrence of $times(event) = (t_1, \ldots, t_n)$ is replaced by $\sigma_c \leq t_1$[3];
- each occurrence of $time(event) = t$ is replaced by $\sigma_c \leq t$.

[1] We denote by $\mathbb{R}^\perp$ the set of positive, computer representable, real numbers augmented with the special symbol $\perp$, the "undefined" value. The value $\perp$ is used to represent the absence of information, either because it is not yet available or because it does not exist, like the duration of a static object.

[2] In the state of the player these values are denoted, respectively, by b_{id} and e_{id}.

[3] We denote by σ_c the component c of the state $\sigma = \langle c, \delta, \tau \rangle$.

A state satisfies the equality $times(event) = (t_1, \ldots, t_n)$ if it enables the first occurrence of that event, i.e., if the value of its clock is less or equal to t_1. Since the player enters a new state in response to an event, for each time instant t_i, $1 \leq i \leq n$ there exists a state σ^i such that $\sigma^i_c = t_i$.

The triple $\{P\}\ c\ \{Q\}$ can be read as: whenever the evaluation of the SMIL command c starts in a state σ^0 which satisfies the assertion P, i.e., $\sigma^0 \models P$, then it terminates in a state σ^f which satisfies the assertion Q, i.e., $\sigma^f \models Q$. According with this interpretation, the following rule of consequence holds:

$$\text{CONSEQUENCE}\ \frac{P_1 \Rightarrow P\{P\}\,c\,\{Q\}Q \Rightarrow Q_1}{\{P_1\}\,c\,\{Q_1\}}$$

As a general remark, in the triple $\{P\}\ c\ \{Q\}$ the precondition P contains the current time instant of the command c, the natural duration of media items which it defines (if applicable) and the occurrence of events. The postcondition Q contains the definition of the time instants in which the elements defined in c begin and/or end. Media items definitions are evaluated through axioms, while for `par`, `seq` and `excl` container more complex rules are needed.

2.3 Notational conventions

In the following section, we use a number of special notational conventions to introduce the set of inference rules describing the semantics of the SMIL elements.

Table 2 lists a set of abbreviations used for the representation of the SMIL elements. For instance `<cmd c>` stands for any SMIL element with the attribute `id = ''c''`. Moreover, we use the general form `end=''k''` and `dur=''k''` to represent the attributes of an element where the meta-variable `k` is either any of the admitted values for the particular attribute, or the special value `void`. The value `void` represents the absence of that attribute in the element and allows us to introduce only one general rule for each SMIL container. With regards to the attribute `begin`, we assume it is always defined since its absence can be represented by the value `k=''0''`. For instance, `<video id=''v''begin=''0''dur=''5''end=''void''/>` is considered as a synonymous of `<video id=''v''dur=''5''/>`.

The advantage of this representation is that of avoiding repetition of very similar rules, but we need a set of predicates to check the existence of an attribute's value before using it. We need also to classify the elements which occur in an SMIL document with respect to the values of their attributes `dur` and `end`. Hence we introduce some auxiliary predicates and sets whose description can be found in Table 4.

Table 4 List of predicates and sets used in the definition of the proof rules

Name	Description
finite(k)	holds if k is a real value
indefinite(k)	holds if k is equal to `''indefinite''`
defined(k)	holds if k is not equal to `''void''`
NotDur	contains all the statements withattributes `dur` and `end` equal to `void`
Closure(`c`)	contains `c` and all the statements defined inside the element `c`, at any level of nesting
Indef(`c`)	holds if in *Closure*(`c`) there are elements with attribute `end` (or `dur`) equal to `''indefinite''`

3 Inference rules for SMIL language

The SMIL language definition provided by Bulterman et al. [8, 9] does not contain a formal presentation of the semantics of elements and attributes. SMIL recommendation is divided into sections, some of which are defined "normative". To clarify the meaning of particular values for attributes, an algorithm is provided to better explain how significant time instants are computed in that case, but neither a formal definition nor verification tools have been implemented by Synchronized Multimedia Working Group of W3C to check the semantic correctness of SMIL elements.

In this section, we define a formal system which is able to find out temporal conflicts of a multimedia presentation defined using SMIL. The system provides a Hoare-like logic for SMIL by a set of inference rules describing how the execution of a piece of code changes the state of the playback. Since the SMIL specification lacks a formal operational semantics, the soundness and completeness of our approach cannot be formally proved, but we consider that our semantics is correct according to any operational semantics which formalizes the changes in the state of a player described in the SMIL recommendation.

Our rules aim at discovering semantic conflicts which can be the cause of unexpected behaviors even if they are often not considered errors by SMIL recommendation. Therefore, our rules may not validate multimedia presentations which contain some conflicts, even if they are correct according to the standard specification. We note here that we do not restrict the set of behaviors which can be described through SMIL, but only the set of documents which describe those behaviors, thus helping the user to develop robust and clear code. In fact, the presence of conflicts, besides being the main cause of unexpected behavior [11, 18], greatly affects the future maintainability of the code.

Table 5 Proof rules for media items definitions

STATIC+BEGIN
$\{A \cup Pre\}$ `<static m begin=''k1''/>` $\{A \cup Post\}$
where $Pre = \{t_{cr}(\mathtt{m}) = start - k1\}$ and $Post = \{begin(\mathtt{m}) = start\}$

CONT+BEGIN
$\{A \cup Pre\}$ `<cont m begin=''k1''/>` $\{A \cup Post\}$
where $Pre = \{t_{cr}(\mathtt{m}) = start - k1, dur(\mathtt{m}) = stop - start\}$
$Post = \{begin(\mathtt{m}) = start, end(\mathtt{m}) = stop\}$

MEDIA+BEGIN+END+DUR
$\{A \cup Pre\}$ `<media m begin='k1' end='k2' dur='k3'/>` $\{A \cup Post\}$
where $Pre = \{t_{cr}(\mathtt{m}) = start - k1\}, Post = \{begin(\mathtt{m}) = start\} \cup End$ and

$$End = \begin{cases} \{end(\mathtt{m}) = start - k1 + k2\} & \text{if } finite(k2) \\ \{end(\mathtt{m}) = start + k3\} & \text{if } finite(k3) \\ \emptyset & \text{otherwise} \end{cases}$$

APPLICABILITY CONDITION:
$(defined(k2) \vee defined(k3)) \wedge ((finite(k2) \wedge finite(k3)) \implies k3 = k2 - k1)$
$\wedge\ (finite(k2) \implies \neg indefinite(k3)) \wedge (finite(k3) \implies \neg indefinite(k2))$

SMIL SPECIFICATIONS [9]
`begin`: defines when the element becomes active;
`end`: describes the end value as an offset from an implicit syncbase;
`dur`: specifies the simple duration.

We start by considering self-contained elements, i.e., SMIL commands whose synchronization does not refer to other media items or elements. The axioms which verify the correctness of statements defining media items are listed in Table 5. The use of interactive events is discussed in Sects. 3.2 and 3.3.

Assume, we want to verify the triple:

$\{P\}$`<video id=''v'' begin=''2''/>`$\{Q\}$

where the precondition P is $\{dur(\mathtt{v}) = 5, t_{cr}(\mathtt{v}) = 0\}$ and the postcondition Q is $\{begin(\mathtt{v}) = 2, end(\mathtt{v}) = 7\}$. We can prove its correctness since we can instantiate the axiom CONT+BEGIN using the values $start = 2$, $k1 = 2$ and $stop = 7$.

The system can also be used to find the begin and end times of media items, i.e., as the basis for the implementation of a player. In this case, the axiom CONT+BEGIN describes the transformation of the state of the player: if the player starts in a state $\sigma^0 \models P$, i.e., $\sigma^0 = \langle 0, \sigma_\delta, \sigma_\tau \rangle$ such that $\sigma_\delta(\mathtt{v}) = \langle \bot, \bot, 5 \rangle$, our rules show that it ends in a state $\sigma^f \models Q$, i.e., $\sigma^f = \langle 7, \sigma^f_\delta, \sigma^f_\tau \rangle$ such that $\sigma^f_\delta(\mathtt{v}) = \langle 2, 7, 5 \rangle$. Thus, we get the values used to start and stop the video.

Our rules allow us to describe also elements where media items are terminated before their natural duration due to the definition of an `end`, or `dur`, attribute. As an example, consider the triple:

$\{t_{cr}(\mathtt{v}) = 0, dur(\mathtt{v}) = 5\}$`<video id=''v'' begin=''2'' end=''3'' dur='void'/>`$\{Q\}$

where $\{Q\} = \{begin(\mathtt{v}) = 2, end(\mathtt{v}) = 3\}$.

We can instantiate the rule MEDIA+BEGIN+END+DUR with the values $start = 2$, $k1 = 2$, $k2 = 3$ and $k3$=`void`. The applicability condition holds since $k2$ is finite, and $k3 =$ `void`. In this case, according to the SMIL recommendation, if the player starts in a state σ which satisfies the precondition, i.e., such that $\sigma_c = 0$, then the final state reached by the player is $\sigma^f = \langle 3, \sigma^f_\delta, \sigma^f_\tau \rangle$ where $\sigma^f_\delta(\mathtt{v}) = \langle 2, 3, 5 \rangle$ and thus $\sigma^f \models Q$.[4]

Note, in Table 5, that the rule MEDIA+BEGIN+END+DUR defines the end time of a media item `m` only if both $k2$ and $k3$ are not equal to "`indefinite`". Moreover, the definition of media items does not lead to temporal conflicts unless the author defines both the `dur` and the `end` attributes. The applicability condition does not allow the application of the rule in presence of incorrect values of these attributes; for instance, when both the attributes `dur` and `end` are finite, the relation $k3 = k2 - k1$ must hold; otherwise, the applicability condition points out the temporal conflict to the user.

3.1 Rules for parallel and sequential composition

When media definitions are nested into parallel and sequential composition, the evaluation of these structures requires the definition of more complex rules.

Because the flexibility of SMIL elements allows us to describe the same temporal behavior using both a `par` or a `seq` element, we base the discussion of this section mainly on the description of the rules for the parallel composition.

[4] According to Definition 1, the third component of $\sigma^f_\delta(\mathtt{v})$ describes the natural duration of `v` and not the actual time for which it remains active on the user screen.

Table 6 Proof rule for the parallel composition when the attribute dur is equal to void

PAR+BEGIN+END

$$\frac{\{A_i \cup \{t_{cr}(\mathtt{c}_i) = init + k1\}\}\ \mathtt{c}_i\ \{B'_i\} \qquad \forall i\ 1 \le i \le n}{\{A'\}\ \texttt{<par c begin='k1' end='k2' dur='void'>}\ \mathtt{c}_1 \ldots \mathtt{c}_n\ \texttt{</par>}\ \{B\}}$$

where

$$A' = \bigcup_{i=1}^{n} A_i \cup \{t_{cr}(\mathtt{c}) = init\}$$

$$B = \bigcup_{i=1}^{n} B_i \cup \{begin(\mathtt{c}) = init + k1\} \cup End$$

$$stop = \begin{cases} init + k2 & \text{if } finite(k2) \\ max_{\mathtt{c}_i}\ \{end_{B_i}(\mathtt{c}_i)\} & \text{if } \neg defined(k2) \end{cases}$$

$$End = \begin{cases} \{end(\mathtt{c}) = stop\} & \text{if } \neg Indef(\mathtt{c}) \\ \emptyset & \text{otherwise} \end{cases}$$

$$B'_i = \begin{cases} B_i \setminus \{end(\mathtt{c}_i) = stop\} & \text{if } \mathtt{c}_i \in NotDur \wedge finite(k2) \\ B_i & \text{otherwise} \end{cases}$$

APPLICABILITY CONDITION:

$finite(k2) \implies ((\neg Indef(\mathtt{c}) \wedge k2 \ge k1) \quad \wedge\ \forall \mathtt{c}_i\ (end_B(\mathtt{c}_i) \le stop \vee \mathtt{c}_i \in NotDur))$
$\wedge\ Indef(c) \implies (\neg defined(k2) \vee indefinite(k2))$
$\wedge\ \forall \mathtt{c}_i\ begin_B(\mathtt{c}_i) \ge init + k1$

SMIL SPECIFICATIONS [9]
A par container, short for "parallel", defines a simple time grouping in which multiple elements can play back at the same time. The implicit syncbase of the child elements of a par is the begin of the par. [...] The implicit duration ends with the last active end of the child elements.

The sequential composition is discussed at the end of this section.

We start our analysis by considering the parallel composition expressed by the element `par` when the attribute `dur` is not present (i.e., `dur=''void''`), the attribute `begin` is present (possibly with zero value) and the attribute `end` is `void`, `indefinite` or holds a real number. The PAR+BEGIN+END rule described in Table 6 defines the semantics of the parallel composition in these cases. In the postcondition, we distinguish the components $B_1 \ldots B_n$ to make it clear that the postcondition contains information about each $\mathtt{c}_i$, be it a media object or a synchronization structure.

To prove the correctness of the element <`par c`> $\mathtt{c}_1 \ldots \mathtt{c}_n$ </`par`>, each $\mathtt{c}_i$ must be proved to be correct by assuming as its current time instant, the current time instant of the parallel element plus the offset given by the attribute `begin` , i.e., if $(t_{cr}(\mathtt{c}) = init)$ is contained into the precondition of the element `c`, the precondition of each element $\mathtt{c}_i$ must contain $(t_{cr}(\mathtt{c}_i) = init + k1)$ where $k1 \ge 0$ is the value of the attribute `begin` and *init* is the time instant at which the statement `par` is evaluated.

The evaluation of the end time instant of a `par` element is not always possible since it is not possible to calculate the end time of a media element in two cases: if it is a static object without an attribute `end` or `dur` defined, or if one of these attributes is equal to `''indefinite''`. In the same way, the ending time of a `par` statement cannot be calculated if its attribute `end` (or `dur`) is equal to `''indefinite''`, or if it contains a child with this value for the attribute `end` (or `dur`). In these cases, the element ends together with the overall presentation.

Once we are able to decide whether a parallel composition terminates, we must calculate the time instant *stop*. If the element `c` does not contain the definition of attribute `end` (i.e., `end = ''void''`) then `c` ends when all its children, which are not static objects, end their playbacks, i.e., at time instant $stop = max_{\mathtt{c}_i}\{end_{B_i}(\mathtt{c}_i)\}$. Otherwise the element `c` ends at time instant $stop = init + k2$, with one exception: we do not assign a stop value when some items defined inside `c` have a longer duration than `c`, defined with an attribute `dur` or `end`. In this case, the author gives a double, and contradictory, definition of the duration of the involved elements, thus generating a temporal conflict. Note that there is a temporal conflict even when the parallel composition has a finite duration, but contains some children with an indefinite duration, which is, by definition, longer than any other finite value.

The applicability condition prevents us to apply the PAR+BEGIN+END rule in these cases. In particular, the condition $finite(k2) \Rightarrow \neg Indef(\mathtt{c}) \wedge k2 \ge k1$ states that in presence of a finite value of $k2$, the rule can be applied to the statement `c` only if it does not contain, at any level of nesting, an item with an indefinite duration. Moreover, the element must end after its beginning, i.e., $k2 \ge k1$. The applicability condition $Indef(\mathtt{c}) \Rightarrow (\neg defined(k2) \vee indefinite(k2))$ states that the attribute `end` must be equal to `''indefinite''`, or `''void''` if the statement does not end. Finally, the condition $\forall \mathtt{c}_i begin_B(\mathtt{c}_i) \ge init + k1$ expresses the fact that all children of `c` must start together with `c` or after it.

Table 7 Proof rules for a general composition of elements when the attribute dur is defined

CMD+BEGIN+END+DUR

$$\frac{\{A\}\ \texttt{<cmd c begin='k1' end='k2' dur='void'>}\ \texttt{c}_1 \ldots \texttt{c}_n\ \texttt{</cmd>}\ \{B\}}{\{A\}\ \texttt{<cmd c begin='k1' end='k4' dur ='k3'>}\ \texttt{c}_1 \ldots \texttt{c}_n\ \texttt{</cmd>}\ \{B\}}$$

APPLICABILITY CONDITION:
$finite(k3) \implies (finite(k2) \wedge k3 = k2 - k1)$
$\wedge\ (defined(k4) \implies k4 = k2) \quad \wedge\ (indefinite(k2) \iff indefinite(k3))$

SMIL SPECIFICATIONS [9]
The attribute dur specifies the simple duration. If the element does not have a (valid) dur attribute, the simple duration [...] is defined to be the implicit duration of the element.

Let us illustrate with a simple example how our rules find out these conflicts. Let us consider the following parallel composition:

```
<par id="p" begin="0" end="5">
    <img id="i" begin="0" end="5" />
    <text id="tx" begin="0" end="7" />
</par>
```

Even if the temporal conflict is evident since the element is simple (text page tx lasts longer than the element in which it is contained), we try to check the semantic correctness of this statement to show how the system works.

We would like to prove that

$$\{t_{cr}(\texttt{p}) = 0\}\texttt{<par p ...>}\{Q\}$$

where $Q \equiv \{begin(\texttt{i}) = 0, end(\texttt{i}) \leq 5, begin(\texttt{tx}) = 0, end(\texttt{t}) \leq 5, begin(\texttt{p}) = 0, end(\texttt{p}) = 5\}$ but statement p is not correct since rule PAR+BEGIN+END (see Table 6) cannot be applied. In fact, since both tx and i do not belong to the set *NotDur*, to apply the rule we would have to prove the premises:

$$S_{\texttt{i}} \equiv \{t_{cr}(\texttt{i}) = 0\}\texttt{i}\{begin(\texttt{i}) = 0, end(\texttt{i}) = 5\}$$
$$S_{\texttt{tx}} \equiv \{t_{cr}(\texttt{tx}) = 0\}\texttt{tx}\{begin(\texttt{tx}) = 0, end(\texttt{tx}) = 5\}$$

The first triple $S_{\texttt{i}}$ is valid and we can prove it by the axiom MEDIA+BEGIN+END +DUR, but we cannot prove the triple $S_{\texttt{tx}}$ which is not valid. Therefore the PAR+BEGIN+ END rule cannot be applied since the premise $S_{\texttt{tx}}$ cannot be verified. In this case, the answer of our tool is that the presentation contains a semantic conflict since media item tx ends at time instant 7 while its father ends at time instant 5 (see Fig. 2).

The rule which describes the semantics of the sequential composition is very similar to the PAR+BEGIN+END rule since the two elements can express the same synchronization if the values of the attributes are properly defined. There are only two differences: first, the current time instant of each child is equal to the end time instant of the previous child, and not to the current time instant of the seq element. Second, the seq statement imposes a duration equal to zero to static media items which have not a defined duration, i.e., $begin_B(\texttt{c}_i) = handend_B(\texttt{c}_i) = hif\,\texttt{c}_i$ is a static media contained in *NotDur*. This means that they are never played in the user screen. In this case, our tool displays a warning to the user which can be considered or not. The complete definition of the rule for the seq statement can be found in [4].

So far we considered only the use of attributes begin and end, but, as already discussed for media item definition, statements can also contain an attribute dur whose semantics is very similar to the end attribute and therefore an easy translation can be obtained with the rule CMD+BEGIN+END+ DUR illustrated in Table 7.

3.2 User interactions in the attributes begin and end

SMIL language permits also the use of events as possible values for the attributes begin and end of the elements (see Table 2). Let us consider first the case in which the start (or the end) of a media, or a group of media items, occurs as a result of an user interaction, e.g. when the user keys in a character, say ‘s’, in the keyboard as described by the following element:

```
<cmd c begin=''accesskey(s)+k''>...</cmd>
```

where accesskey(s) means that the user has to key in the character ‘s’ and $k \geq 0$ represents a number of seconds.

The correctness of this statement can be proven only if we already know the instant in which the event accesskey(s) takes place. According to Sect. 2.1, the player recorded the last occurrence of an interactive event in the state through the function σ_τ which records the time instant in which an event occurs. Since the player enters a new state in response to an event, for each occurrence of an event, there exists a state such that the clock value is equal to the time instant recorded from the function σ_τ. An interactive event may involve a single media item, or the global environment, as in the case of a digit on the keyboard. In the precondition of a statement, we use the

Table 8 Proof rules for SMIL statements with an interactive event in the definition of the `begin` or the `end` attribute

BEGIN+USER-INTERACTION

$$\frac{\{A \cup \{t_{cr}(\texttt{c}) = n\}\}\texttt{<cmd c begin=`k1' end=`k2'>...</cmd>}\{B\}}{\{A'\}\texttt{<cmd c begin=`event+k1'end=`k2'>...</cmd>}\{B\}}$$

where $A' \overset{def}{=} A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{time(event) = n\}$
APPLICABILITY CONDITION:
$\{A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{time(event) = n\}\} \implies \{n \geq init\}$

END+USER-INTERACTION

$$\frac{\{A \cup \{t_{cr}(\texttt{c}) = init\}\}\texttt{<cmd c begin=`k1'>...</cmd>}\{B \setminus \{end(\texttt{c}) = n + k2\}\}}{\{A'\}\texttt{<cmd c begin=`k1' end=`event+k2'>...</cmd>}\{B\}}$$

where $A' \overset{def}{=} A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{time(event) = n\}$
APPLICABILITY CONDITION:
$\{B \cup \{time(event) = n\}\} \implies \{n \geq begin_B(\texttt{c})\} \cup \{end_B(\texttt{c}) = n + k2\}$

SMIL SPECIFICATIONS [9]
attribute **begin**: Describes an event and an optional offset that determine the element begin.
The element begin is defined relative to the time that the event is raised. Events may be any event defined for the host language [...] and may include user-interface events, event-triggers transmitted via a network, etc.
attribute **end**: Describes an event and an optional offset that determine the end value. The end value is defined relative to the time that the event is raised. Events may be any event defined for the host language [...] and may include user-interface events, event-triggers transmitted via a network, etc.

functions *times*(*event*) and *time*(*event*), to constraint the input events. For instance the precondition $\{time(event) = n\}$ states that the initial state in which the element is evaluated should enable the event *event*, i.e., its clock should be $\leq n$. The occurrence of the event changes the current time instant of the element, which is now equal to the time instant in which the event takes place.

Table 8 shows the rules to deal with statements with a `begin` or an `end` attribute which is bound to an interactive event. Let us consider again the example of an input from the keyboard:

$\{A'\}$`<cmd c begin=‘‘accesskey(s)+k1’’ end=‘‘k2’’>...</cmd>`$\{B\}$

where $A' = \{A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{time(\texttt{accesskey(s)}) = keyin\}\}$.

These rules state that the element must be evaluated with reference to the time instant in which the event occurs, i.e., if $(time(\texttt{accesskey(s)}) = keyin) \in A'$, we can prove the correctness of the element `c` if we can prove that

$\{A \cup \{t_{cr}(\texttt{c}) = keyin\}\}$`<cmd c begin=‘‘k1’’end=‘‘k2’’>...</cmd>`$\{B\}$

holds. The input from the keyboard must occur after the evaluation of the statement, represented by the value *init*, or after its beginning if `accesskey` is defined in the `end` attribute of a statement.

The correctness of this rule derives from the following considerations: consider an initial state σ^0 such that the precondition A' holds: $\sigma^0 \models \{A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{time(\texttt{accesskey(s)}) = keyin\}\}$. Therefore, $\sigma^0{}_c = init$, $init \leq keyin$ and it exists an intermediate state γ such that $\gamma_c = keyin$ and $\gamma_\tau(\texttt{accesskey(s)}) = keyin$. Hence, by monotonicity, $\gamma \models \{A \cup \{t_{cr}(\texttt{c}) = keyin\}\}$ and then, by the correctness of the premises, we have that the player reaches a final state σ^f which satisfies the postcondition B. In this particular case, $\{begin(\texttt{c}) = keyin + k1\} \subset B$, which means that `c` starts exactly $k1$ time instants after the occurrence of the event `accesskey(s)`, i.e., our rule respects the standard specifications (see Table 8).

Note that all other interactive events supported by the SMIL specifications (a partial list can be find in Table 2) could be addressed in the same way since the player records the time instant in which the event occurs by means of the function σ_τ. As an example, the `activateEvent` represents the time instant in which a user clicks on a media item and, from our point of view, it is not different from the user clicking on the keyboard. Also, in this case, the only constraint is that the event must occur after the evaluation of the statement to be useful.

An interactive event may occur more than once, e.g., the user may click several times on a button. This means that an object which binds its start to this event may play more than once. As discussed in Sect. 2.1 to record all the

Table 9 Proof rule for multiple executions of the same element

BEGIN+MULTIPLE-PLAYBACKS

$$\frac{\forall i 0 \leq i \leq n \{A \cup \{t_{cr}(\nu^i(\texttt{c})) = start_i\}\}\texttt{<cmd } \nu^i(\texttt{c}) \texttt{ begin='k1' end='k2'>...</cmd>}\{B'_i\}}{\{A'\}\ \texttt{<f><cmd c begin='event+k1' end='k2'>...</cmd></f>}\ \{\cup_{i=1}^{n} B_i\}}$$

where $\texttt{f} \in \{\texttt{par}, \texttt{excl}\}$

$A' \overset{def}{=} A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{sequence\}$

$sequence \overset{def}{=} times(event) = (start_0, \ldots, start_n)$

$$B'_i = \begin{cases} B_i \setminus \{end(\nu^i(\texttt{c})) = h\} & \text{if } (begin_B(\nu^{i+1}(\texttt{c})) = h) \\ B_i & \text{otherwise} \end{cases}$$

APPLICABILITY CONDITION:

$\forall(i,j)\ i \leq j \implies (start_i \leq start_j)$
$\wedge\ \{A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{sequence\}\} \implies \forall i\ (start_i \geq t_{cr}(\texttt{c}))$
$\wedge\ \forall(i,j)\ (begin_B(\nu^i(\texttt{c})) \leq begin_B(\nu^j(\texttt{c}))) \implies (end_B(\nu^i(\texttt{c})) \leq begin_B(\nu^j(\texttt{c})))$

SMIL SPECIFICATIONS [9]
event-values, accesskey-values [...] do not yield an instance time unless and until the associated event happens. Each time the event happens, the condition yields a single instance time. The event time plus or minus offset is added to the list. If the event happens multiple times during a parent simple duration, there may be multiple instance times in the list associated with the event condition.

executions of an item, we consider multiple playbacks of the same item as new objects with new names. The rule BEGIN+MULTIPLE-PLAYBACKS[5] (see Table 9) models this situation: multiple executions of the same item are correct as soon as each single playback is correct. We note that if an event occurs before the end of the media then it is immediately stopped and restarted.[6] In this case, we do not need to show the end time instant of the playback in the premises as stated by the rule.

Let us consider the previous example where the user keys in the digit ‘s’ twice, at time instants $start_1 < start_2$ and let $A' = \{A \cup \{times(\texttt{accesskey(s)}) = (start_1, start_2)\}\}$. We can apply the rule BEGIN+MULTIPLE-PLAYBACKS to prove

$$\{A'\}\texttt{<cmd c begin=``accesskey(s)+k1'' end=``k2''>...</cmd>}\{B\}$$

if we can prove the correctness of both the executions of the element c

$$\{A_1\}\texttt{<cmd c begin=``accesskey(s)+k1'' end=``k2''>...</cmd>}\{B'_1\}$$
$$\{A_2\}\texttt{<cmd } \nu(\texttt{c}) \texttt{ begin=``accesskey(s)+k1'' end=``k2''>...</cmd>}\{B'_2\}$$

where $v(\texttt{c})$ is a new name for the second execution of c, $A_1 = \{A \cup \{t_{cr}(\texttt{c}) = start_1\}\}$, $A_2 = \{A \cup \{t_{cr}(v(\texttt{c})) = start_2\}\}$.

If the interactive event appears in the definition of the end attribute, we need different considerations. Differently from the case of an event defined in the begin attribute, multiple occurrences of an event defined in the end attribute do not cause multiple playbacks of a element: the player will end the element as soon as the ending event occurs for the first time, subsequent occurrences of the same event are ignored. The rule END+USER-INTERACTION (Table 8) can be used in this case, since any state which satisfies the first element of a sequence of event times satisfies, by definition, also the sequence. The correctness of the rule derives from the fact that subsequent occurrences of the same event must be ignored.

A particular case regards media items which contain interactive events both in the begin and in the end attributes. Rule BEGIN+END+MULTIPLE-PLAYBACKS (see Table 10) extends the case when the begin attribute has a unique event, and could be used to prove the correctness of each single execution of the item. The only relevant differences are contained in the applicability condition: the two sequences must be ordered and the sequence of starts must have a length equal or greater than the sequence of ends. Moreover, the last event must be a stop event.

3.3 Management of non interactive events

Another possibility offered by the SMIL standard is to bind the begin (or end) event of a (group of) media item m with the begin (or end) event of another (group of) media item n. As already discussed in Sect. 2.1, non-interactive events are recorded differently in the state of the player, since they are not traced by the function σ_τ but by function σ_δ. As an example, consider the elements

```
<par id="p" end="au.end">
    <audio id="au" />
    <text id="tx" />
</par>
```
(1)

```
<cmd id="m" begin="n.begin+5"/>
```
(2)

in case (1), the whole par statement ends when media item au ends; in case (2) media item m begins 5 s after the beginning of n.

[5] We consider only positive offsets since the rule deals only with interactive events.

[6] We suppose that the attribute restart assumes the default value always.

Table 10 Proof rule for media items which contain an interactive events both in the `begin` and in the `end` attributes

BEGIN+END+MULTIPLE-PLAYBACKS

$$\frac{\{A'\}\ \texttt{<cmd c begin='k1'>...</cmd>}\{B_1'\} \qquad n>1 \implies \{A\text{"}\}\ \nu(\texttt{c})\ \{B\text{"}\}}{\{A''\}\texttt{<cmd c begin='event1+k1' end='event2+k2'>...</cmd>}\{B\}}$$

where

$$\begin{array}{ll}
A' & \overset{def}{=} A \cup \{t_{cr}(\texttt{c}) = start_1\} \\
A'' & \overset{def}{=} A \cup \{t_{cr}(\texttt{c}) = init\} \cup \{starts\} \cup \{stops\} \\
starts & \overset{def}{=} times(event_1) = (start_1, \ldots, start_n) \\
stops & \overset{def}{=} times(event_2) = (stop_1, \ldots, stop_m) \\
B & \overset{def}{=} B_1 \cup B\text{"} \\
B_1' = & \begin{cases} B_1 \setminus \{end(\texttt{c}) = h\} & \text{if } (begin_B(\nu(\texttt{c})) = h) \\ B_1 & \text{otherwise} \end{cases} \\
starts' & \overset{def}{=} times(event_1) = (start_2, \ldots, start_n) \\
stops' & \overset{def}{=} \begin{cases} times(event_2) = (stop_2, \ldots, stop_m) & \text{if } (end_B(\texttt{c}) = stop_1 + k2) \\ stops & \text{otherwise} \end{cases} \\
A\text{"} & = A \cup \{t_{cr}(\nu(\texttt{c})) = init\} \cup \{starts'\} \cup \{stops'\}
\end{array}$$

APPLICABILITY CONDITION:

$m \leq n\ \wedge (\forall(i,j)\ i \leq j \implies start_i \leq start_j)\ \wedge (\forall(i,j)\ i \leq j \implies stop_i \leq stop_j)$
$\wedge\ A' \cup B \implies (stop_1 \geq begin_B(\texttt{c})$
$\wedge((end_B(\texttt{c}) = stop_1 + k2) \vee (end_B(\texttt{c}) = begin_B(\nu(\texttt{c})))))$

SMIL SPECIFICATIONS [9]
event-values, accesskey-values [...] do not yield an instance time unless and until the associated event happens. Each time the event happens, the condition yields a single instance time. The event time plus or minus any offset is added to the list. If the event happens multiple times during a parent simple duration, there may be multiple instance times in the list associated with the event condition.

Element (1) can be treated similarly to the case of interactive events: if we already know (from the premise) the end point of `au`, i.e., `end(au) = stop`, then we can analyze the element `<par id=''p'' end=''-stop''>...</par>`. Therefore, the following rule can be applied to case (1).

PAR+END+EVENT

$$\frac{\{A\}\texttt{<par c end='stop} \pm \texttt{k'>c}_1\texttt{..c}_n\texttt{</par>}\{B\}}{\{A\}\texttt{<par c end='ci.end} \pm \texttt{k'>c}_1\texttt{..c}_n\texttt{</par>}\{B\}}$$

APPLICABILITY CONDITION:
$end_B(\texttt{c}_i) = \texttt{stop} \wedge begin_B(\texttt{c}) \leq stop \pm k$

The situation is more complex in case (2), which cannot be analyzed singularly since its evaluation needs information about the begin attribute of media item `n`. For this reason, we must consider a set of media items as shown by the following rule:

BEGIN+EVENT

$$\frac{\{A\}\texttt{<cmd c>}\ldots\texttt{c}_i\ldots\texttt{c}_j'\ldots\texttt{</cmd>}\{B\}}{\{A\}\texttt{<cmd c>}\ldots\texttt{c}_i\ldots\texttt{c}_j\ldots\texttt{</cmd>}\{B\}}$$

where c_i, c_j and c'_j are related as follows: there exist *n*, *m* such that

$\texttt{n} \in Closure(\texttt{c}_i)$,

$\texttt{m} \overset{def}{=} \texttt{<cmd id="m" begin="n.begin+k">...</cmd>} \in Closure(\texttt{c}_j)$,

$\texttt{c}_j$ is obtained from $\texttt{c}_j$ by replacing `m` with

$\texttt{<cmd m begin="begin}_B\texttt{(n)+k">...</cmd>}$.

A particular attention is required if there are multiple executions of the media item `n` due to user interactions. For example, consider a video `v` activated by a click on an image `i` and its associated soundtrack `a`:

```
<par>
    <img id = "i" />
    <video id = "v" begin="i.activateEvent"/>
    <audio id = "a" begin="v.begin"/>
</par>
```

In this case, even the soundtrack `a` must be played several times. If we try to prove the correctness of this parallel composition, we need to analyze any single component. In particular, by applying the rule BEGIN+MULTIPLE-PLAYBACKS to video `v`, we obtain the set $\{\texttt{v}, \nu(\texttt{v}), \ldots \nu^h(\texttt{v})\}$ of activations of the same video `v`, but we have to consider also the

Table 11 Proof rule for SMIL elements containing multiple values in the `begin` attribute

CMD+BEGIN-VALUE-LIST+END

$$\frac{finite(\texttt{k1}) \Rightarrow \{A'\}\ \texttt{c}_{k1}\{B'_{k1}\} \qquad \forall i\ \{A \cup \{t_{cr}(\nu^i(\texttt{c})) = init\}\}\ \nu^i(\texttt{c})\ \{B'_i\}}{\{A''\}\ \texttt{<f><cmd c begin='event-list' end='k2' dur='void'>...</cmd></f>}\ \{B\}}$$

where $\texttt{f} \in \{\texttt{par}, \texttt{excl}\}$

$\texttt{event-list} \overset{def}{=} event_1; \ldots; event_n; \texttt{k1}$

$(\forall i)\ \nu^i(\texttt{c}) \overset{def}{=}$ `<cmd` $\nu^i(\texttt{c})$ `begin='`$event_i$`' end='k2' dur='void'>...</cmd>`

$\texttt{c}_{k1} \overset{def}{=}$ `<cmd` $\texttt{c}_{k1}$ `begin='k1' end='k2' dur='void'>...</cmd>`

$A' \overset{def}{=} A \cup \{t_{cr}(\texttt{c}_{k1}) = init\}$

$A'' = A \cup \{t_{cr}(\texttt{c}) = init\}$

$$B = \begin{cases} \cup_1^n B_i \cup B_{k1} & \text{if } finite(\texttt{k1}) \\ \cup_1^n B_i & \text{otherwise} \end{cases}$$

$$B'_i = \begin{cases} B_i \setminus \{end(\nu^i(\texttt{c})) = h\} & \text{if } \exists j\ (begin_B(\nu^j(\texttt{c})) = h) \\ B_i & \text{otherwise} \end{cases}$$

$$B'_{k1} = \begin{cases} B_{k1} \setminus \{end(\texttt{c}_{k1}) = h\} & \text{if } \exists j\ (begin_B(\nu^j(\texttt{c})) = h) \\ B_{k1} & \text{otherwise} \end{cases}$$

APPLICABILITY CONDITION:

$finite(k1) \vee (\exists i\ time(event_i) = n_i \wedge finite(n_i))$

SMIL SPECIFICATIONS [9]

A semi-colon separated list of begin values. [...] In general, the earliest time in the list determines the begin time of the element. [...] Each element can have a begin attribute that defines one or more conditions that can begin the element. [...] In order to calculate the times that should be used for a given interval of the element, we must convert the begin times and the end times into parent simple time, sort each list of times (independently), and then find an appropriate pair of times to define an interval.

set of associated executions of a, $\{\texttt{a}, \ldots, \nu^h(\texttt{a})\}$. To obtain that, we must refine the BEGIN+EVENT rule by requiring that $\texttt{c}'_j$ is obtained from $\texttt{c}_j$ by expanding `m` with all its executions $\nu^i(\texttt{m})$. Each

$\nu^i(\texttt{m}) \overset{def}{=}$ `<cmd` $\nu^i(\texttt{m})$ `begin="begin`$_B(\nu^i(\texttt{n}))$`+k">...</cmd>`

is associated with a playback of `n`, as stated in the post-condition: $\forall i\{begin(\nu^i(n)) = value_i\} \in B$.

3.4 Multiple values for the attributes `begin` or `end`

The SMIL standard allows us to define an unordered list of value for the attribute `begin` and `end`. This list may contain, separated by a semicolon, a list of events, or a time instant. In this case, the element starts or ends, respectively, as soon as one of the events contained in the list occurs or the player's clock reaches the defined time instant. Then, each time an event in the list occurs (or the time instant is reached), it is restarted. Rule CMD+BEGIN-VALUE-LIST+END (see Table 11) describes the behavior of an element with a list of values for the attributes `begin`. This rule simply states that, if the element `c` can be formally proved to be correct for each possible occurred event in the list, or for the time instant when defined, then it is correct also for the entire list. The postcondition B records all the executions.

3.5 The `excl` element

The SMIL language also provides a container for the *exclusive* composition of media items, i.e., the element `excl`, whose semantics states that only one of its children is active at any given time instant. This is very similar to the sequential composition where only one child is active at any time, but `excl` does not impose any order in the visualization of the children. This means that each child may contain the attribute `begin` in the definition, or may be activated by the user, e.g. following a link. Let us consider the following example:

```
<par>
    <img id = "a" />  <img id = "b" /> <img id = "c" />
    <excl id="e" dur="10">
        <video id = "video_a" begin="a.activateEvent"/>
        <video id = "video_b" begin="b.activateEvent"/>
        <video id = "video_c" begin="c.activateEvent"/>
    </excl>
</par>
```

in this case, the user chooses a video clip by clicking on an image button chosen between media items `a`, `b` and `c`. The corresponding video is activated by the proper *activateEvent*. The `excl` element simply states that only one video clip plays at a time: in fact, the video currently playing is stopped when the user clicks on another image, choosing another video clip.

Table 12 Proof rule for the exclusive composition when the attribute `dur` is equal to `void`

EXCL+BEGIN+END

$$\frac{\{A \cup \{t_{cr}(\mathtt{c}_i) = init + k1\}\}\ \mathtt{c}_i\ \{B'_i\} \qquad \forall i\ 1 \le i \le n}{\{A'\}\ \texttt{<excl c begin='k1' end='k2'dur='void'>}\ \mathtt{c}_1 \ldots \mathtt{c}_n\ \texttt{</excl>}\ \{B\}}$$

where
$$A' = A \cup \{t_{cr}(\mathtt{c}) = init\}$$
$$B = \bigcup_{i=1}^{n} B_i \cup \{begin(\mathtt{c}) = init + k1\} \cup End$$
$$stop = \begin{cases} init + k2 & \text{if } finite(k2) \\ max_{\mathtt{c}_i}\{end_{B_i}(\mathtt{c}_i)\} & \text{if } \neg defined(k2) \end{cases}$$
$$End = \begin{cases} \{end(\mathtt{c}) = stop\} & \text{if } \neg Indef(\mathtt{c}) \\ \emptyset & \text{otherwise} \end{cases}$$
$$B'_i = \begin{cases} B_i \setminus \{end(\mathtt{c}_i) = t_i\} & \text{if } \exists j\ begin_B(\mathtt{c}_j) = t_i \vee (finite(k2) \wedge \mathtt{c}_i \in NotDur) \\ B_i & \text{otherwise} \end{cases}$$

APPLICABILITY CONDITION:
$$finite(k2) \implies \neg Indef(\mathtt{c}) \wedge k2 \ge k1$$
$$\wedge\ Indef(\mathtt{c}) \implies (\neg defined(k2) \vee indefinite(k2))$$
$$\wedge\ finite(k2) \implies \forall i\ (end_B(\mathtt{c}_i) \le stop \vee \mathtt{c}_i \in NotDur)$$
$$\wedge\ \forall(i)\ begin_B(\mathtt{c}_i) \ge init + k1$$
$$\wedge\ \forall(i,j)\ (begin_B(\mathtt{c}_i) \le begin_B(\mathtt{c}_j)) \implies (end_B(\mathtt{c}_i) \le begin_B(\mathtt{c}_j))$$

SMIL SPECIFICATIONS [9]
SMIL 3.0 defines a time container with semantics based upon par, but with the additional constraint that only one child element may play at any given time. If any element begins playing while another is already playing, the element that was playing is stopped. [...] The implicit syncbase of the child elements of the excl is the begin of the excl. The default value of begin for children of excl is "indefinite". This means that the excl has 0 duration unless a child of the excl has been added to the timegraph.

The example shows how the `excl` container does not deal with the activation of its children but with their deactivation; in fact, the playback order of the video clips completely depends on the user choices and not on the elements' definitions.

The semantics of the `excl` element is described in Table 12 by rule EXCL+BEGIN+END. Like elements `par` and `seq`, the `excl` element begins at its current time instant, or after $k1$ time instants if the attribute `begin` is finite, and ends when there are no children playing. This means that, it can have an instantaneous duration if no child starts together with it. For this reason the attribute `end` of this statement usually does not contain the special value ``void''.

The EXCL+BEGIN+END rule is very similar to the rule which describes the semantics of parallel composition, therefore we do not repeat here the problem of the termination of the element. Even in this case, to prove the correctness of the statement `<excl>` $\mathtt{c}_1 \ldots \mathtt{c}_n$ `</excl>` each $\mathtt{c}_i$ must be proven to be correct, assuming the time instant of its parent element as its current time instant. Because the exclusive container may impose a premature stop of the playback of its children, we do not require to know in advance the time instant in which the child $\mathtt{c}_i$ ends:

1. when $\mathtt{c}_i$ ends together with `excl` because it does not contain the attribute `end` or `dur` in its definition (i.e., `k2` is finite and $\mathtt{c}_i \in NotDur$) or
2. when the playback of $\mathtt{c}_i$ is stopped before its termination due to the user interaction or some other external event (i.e., $\exists j | begin_B(\mathtt{c}_j) = t_i$).

The applicability condition prevents the application of the rule in presence of temporal conflicts. Among the conditions already discussed for the parallel composition, the condition $\forall \mathtt{c}_i, \mathtt{c}_j (begin_B(\mathtt{c}_i) \le begin_B(\mathtt{c}_j)) \Rightarrow (end_B(\mathtt{c}_i) \le begin_B(\mathtt{c}_j))$ states that only one child plays at any given time instant, i.e., if child $\mathtt{c}_i$ begins before $\mathtt{c}_j$, it also ends before $\mathtt{c}_j$'s beginning.

We note here that each child can be played several times, since their executions are usually driven by user interaction, e.g., in the previous example, the user may click more than once on the images `a`, `b` and `c`. This situation is solved by applying the rule BEGIN+MULTIPLE-PLAYBACKS (see Table 9) to the repeating media item.

4 Prototype implementation

Based on the formal semantics described in the previous section, we have implemented a tool for the analysis of an SMIL document. Our tool allows us to check consistency during all the authoring phases, whenever the author asks it or saves her/his work.[7]

Section 4.1 describes the system architecture and implementation, and a user test case are reported in Sect. 4.2.

[7] We do not consider dynamic checking a good solution since temporary inconsistencies, due to the *work-in-progress*, should be allowed. This choice is also cost effective.

4.1 The semantic validator

The *semantic validator* implemented has two goals: first, it assists the user in the complex task of authoring a multimedia presentation, automatically finding temporal inconsistencies and helping their correction. Secondly, it produces the sequence of begin and end times of media items contained in an SMIL file to be used for the playback of the presentation. For this reason, our implementation keeps separated the *semantic validator module* (SVM in Fig. 1), which is the engine that can be used both by a *player* and by an *authoring system*, from the interface for the automatic verification of SMIL files. We note again that we do not want to realize a new authoring system (a number of them has been implemented offering different useful facilities like visual editors, preview window, etc.) but a new tool to help the user to discover and fix authoring mistakes, since this facility is still absent.

Figure 1 shows the system architecture: the tool has been implemented to be used in combination with any authoring interface to avoid forcing designers to change their preferences. Moreover, it can be part of an automated pipeline for the generation of SMIL documents. As an example, consider documents automatically build up on the base of the user context. The adaptation process can select different combinations of media items, e.g. a video can be replaced by a set of images due to low network bandwidth. If the document structure is complex, it could be very difficult for the author to consider all the possible choices and their combinations in the overall structure. In this case, our tool can be used to check the correctness of the final result.

If the document is correct, the validation process returns as output the correct sequence of start and stop events of all media items involved in the presentation. This information can be used by a player for the playback of the presentation, or to implement a preview window in an authoring system. We note here that the use of our semantic validator in conjunction with an authoring system is particularly important since the composition of a SMIL document

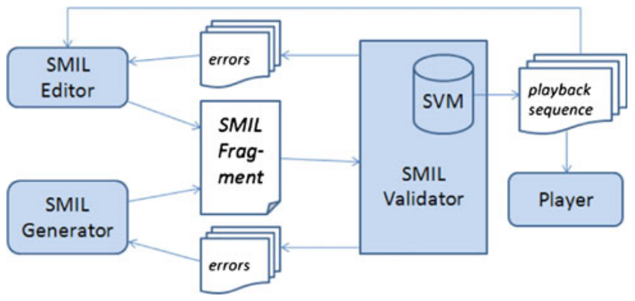


Fig. 1 Architecture of a general authoring system in which our tool can be integrated

driven by the rules is correct by construction. Finally, since the tool does not allow the presence of temporal conflicts, it helps to design robust and clear code, thus improving maintainability of the SMIL documents.

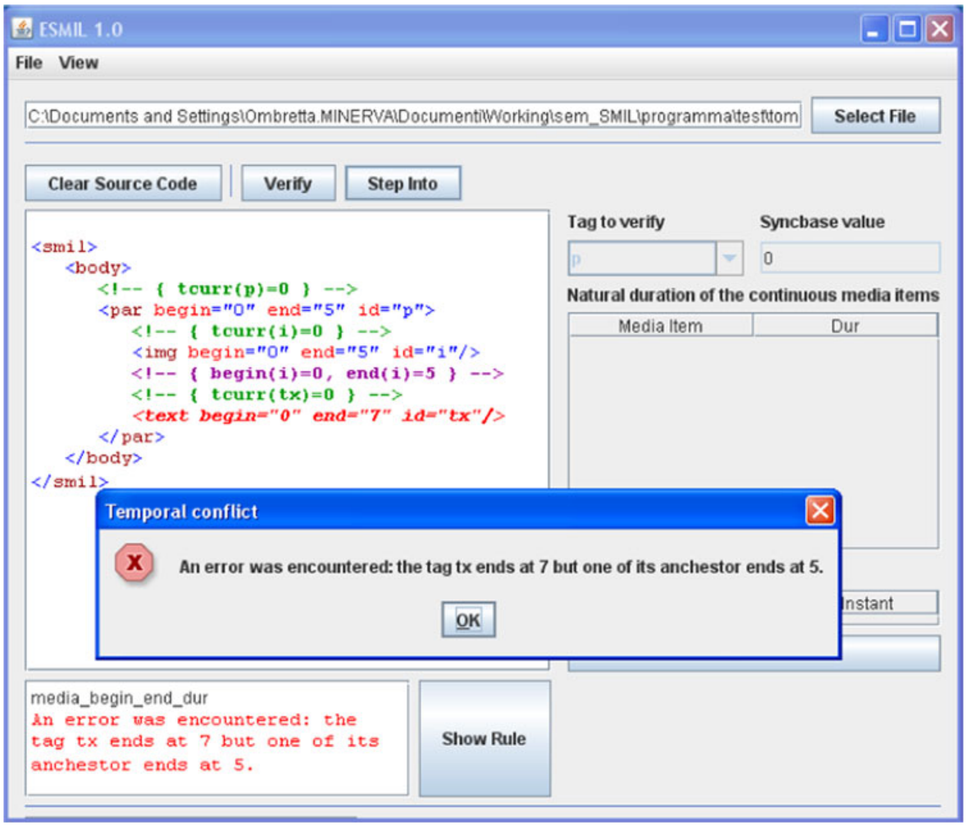
The interface has been implemented using the *Java* language with the goal to test the engine, realized with the *Prolog* language, and to support multimedia authoring by pointing out conflicting values in the document. Figure 2 shows a screenshot of the tool. The user selects an SMIL file to be evaluated and the semantic validator checks its syntactic correctness and displays the code emphasizing elements and attributes. In a second step,[8] the tool asks the user to input the precondition, i.e., the natural duration of continuous objects, the user interactions, and the element to evaluate. The user can ask the analysis of a single selected element or that of the entire document, by selecting the first element of the document; in any case, she/he must give as input the time instant in which the element should be evaluated. Then, the validation process can be initiated and, once concluded, it returns the correct begin and end time of selected element and its content. If any temporal inconsistency is found, the tool prompts a message containing the element with the mistake and its motivation (see Fig. 2). Usually this information is sufficient to help the user to easily detect and correct the error. However, the tool provides also a "step by step" modality by clicking on the "Step Into" button, which shows, for each interaction, a single step of the process, displaying in the source code (respectively before and after the element itself) the precondition and the postcondition of each analyzed element (Fig. 3 shows the final result).

This second modality allows a better understanding of the overall process and of the context in which a single element is evaluated. For example, this is particularly useful in the case of SMIL elements which refer to user interaction in their definition: in this case only events that occur after their evaluation should be considered. Moreover, the evaluation discussed in Sect. 4.2 revealed that this feature helps the designers to clearly state the cause–effect relationships between SMIL elements and the corresponding scheduling sequence for media items. Therefore, it represents an help for non-expert designer to learn the language.

A panel, positioned below the SMIL code, contains the tool's messages to the user, i.e., errors or warnings (e.g., a element with a duration equals to zero), and the last used rule. Expert users can visualize the last used rule by clicking on the button "Show Rule" which shows the corresponding table of this paper.

If the document contains a conflict, the construction of the proof fails because one of the needed premises cannot

[8] The interface buttons are activated/deactivated in order to guide the user to a correct sequence of interactions.



C:\Documents and Settings\Ombretta.MINERVA\Documenti\Working\sem_SMIL\programma\test\tom

```
<smil>
   <body>
      <!-- { tcurr(p)=0 } -->
      <par begin="0" end="5" id="p">
         <!-- { tcurr(i)=0 } -->
         <img begin="0" end="5" id="i"/>
         <!-- { begin(i)=0, end(i)=5 } -->
         <!-- { tcurr(tx)=0 } -->
         <text begin="0" end="7" id="tx"/>
      </par>
   </body>
</smil>
```

Tag to verify: p

Syncbase value: 0

An error was encountered: the tag tx ends at 7 but one of its anchestor ends at 5.

```
media_begin_end_dur
An error was encountered: the
tag tx ends at 7 but one of its
anchestor ends at 5.
```

Fig. 2 In this screenshot eSMIL points out to the user the existence of a temporal conflict between the end time of a text element and the end time of par element containing it

```
<smil>
   <body>
      <!-- { tcurr(p)=0 } -->
      <par begin="0" end="5" id="p">
         <!-- { tcurr(i)=0 } -->
         <img begin="0" end="5" id="i"/>
         <!-- { begin(i)=0, end(i)=5 } -->
         <!-- { tcurr(tx)=0 } -->
         <text begin="0" end="5" id="tx"/>
         <!-- { begin(tx)=0, end(tx)=5 } -->
      </par>
      <!-- { begin(p)=0, end(p)=5, begin(i)=0,
            end(i)=5, begin(tx)=0, end(tx)=5 } -->
   </body>
</smil>
```

Fig. 3 Pre/post condition of an element inserted by eSMIL in the source code

be proved or the applicability conditions are not satisfied. The compositionality of our approach helps the user to correct the mistake and to incrementally continue the analysis.

The *Semantic Validator Module* has been implemented using the Prolog language. This choice allows us to use the Prolog search engine to verify the correctness of an SMIL element. Moreover, since SMIL scripts are XML files, a parser which translates an SMIL script into the text file defining the Prolog database is very easy to implement (e.g. using an XSLT stylesheet).

We associate a fact which describes the type and the attribute `id` to each element: `img(id)`, `text(id)`, `brush(id)`, `audio(id)`, `video(id)`, or `animation(id)`, for media objects, where `id` is the identifier of the object and the chosen predicate depends its type, and `par(id,L)`, `seq(id,L)` and `excl(id,L)`, for synchronization containers where `L` is the list of SMIL elements nested into that container. The list of attributes of an element is described by `attr(id,LoA)` where `LoA` is a list of pairs `[attribute, value]`.

All the semantic rules introduced in the previous sections are translated into Prolog clauses which describe how the precondition is modified to obtain the postcondition. Precondition and postcondition are sets of facts like `begin(id,bm)` and `end(id,em)`, which translate the assertion $begin(id) = bm$ and $end(id) = em$, respectively. The fact `time(event,k)`, used in the precondition, succeeds if the event `event` occurs at time instant `k` and the fact `times(event,L)` succeeds if `L` is the list of time instants in which the event occurs.

The proof rules are implemented using Prolog clauses of two possible types: the first one, `applicable`, translates both the applicability condition for the rules and the constraint due to the structure of the element, and the second one, `apply`, describes how the postcondition is obtained by the applying the rule to the precondition. As an example, let us consider the CONT+BEGIN rule. The Prolog clause:

```
applicable(cont_begin,M):- cont(M),
                           attr(M,[[begin,_]]),
                           tcurr(M,_),dur(M,_).
```

describes the constraints on the structure of the element, in this case it must be a continuous object with an attribute `begin`, and the applicability conditions which are the presence of the facts `tcurr(M,_)` and `dur(M,_)` in the database. The Prolog clause:

```
apply(cont_begin,M):- remove(tcurr(M,N)),
       attr(M,[[begin,N1]]), X is N+N1,
       insert_begin(M,X), dur(M,N2),
       Y is X+N2, insert_end(M,Y).
```

describes how the pre/post condition are modified by the CONT+BEGIN rule. The system removes the fact `tcurr`, so that no other rule can be applied to the same command, and inserts adequate `begin` and `end` facts. As a result, the execution of the prolog program becomes deterministic and thus its complexity is linear with respect to the number of elements in the SMIL script.

This structure allows us to easily add new rules or modify an existing one without affecting the others or to degrade an error to a warning and vice versa.

The correctness of a SMIL script is evaluated by `resolve(c)`. It succeeds if:

- there exists a fact `tcurr(c,_)` and an applicable rule for `c` which succeeds. If `c` is not a media item, it enables the verification of the children by inserting, in the database, a fact `tcurr(ci,t)`, with the suitable value `t` for each `ci` nested in `c`.
- no `tcurr(c,_)` exists. In this case, all the elements have already been evaluated.

If any temporal inconsistency is found, the computation fails and the system returns the element which contains the mistake and gives suggestions on how to correct it.

4.2 Evaluation and test of the prototype

The tool has been developed in an academic environment and has been tested in two ways. First of all, the correctness of the output has been successfully tested with the W3C SMIL Testsuite [10], and provided positive results. Next, we tested and evaluated the effectiveness of our prototype with the help of our students. We assigned to two groups of students of a course on Hypermedia Systems of the second level degree in Computer Science the task of creating a complex SMIL multimedia presentation. Only one group could use our tool during the authoring activity.

The test consisted in the analysis of the performance of the two groups of students. This experience, spreaded over 3 years, has been very positive. First of all, the use of the tool reduced to zero the number of delivered homeworks still containing temporal conflicts: this number ranged from 20 to 30% without our tool. We also noted that the average time needed to produce the final presentation decreased by about 15%. This percentage was calculated for students of average ability, since it obviously decreased for excellent students. Moreover, the code developed by the students improved its quality with respect to the previous years, since we noted that the use of the tool encouraged a better use of the attributes, e.g., it helps to understand that a `begin` attribute in a `par` element is better than to assign the same attribute to all its children. In fact, the code is more maintainable if it contains the minimum number of attributes.

In the last year we also submitted a questionnaire to a group of 10 users, in the range of 20–30 years, to collect their feedbacks. The questionnaire consisted of multiple choice questions which asked to the users theirs degree of expertise in SMIL, the number of conflicts discovered by the tool and the overall experience using it. The users rated the answers on a scale from 0 to 5. A last open question gave the possibility to express additional remarks.

The questionnaires revealed the following insights. The users declared to have a good knowledge of the SMIL language (mean 4.1, standard deviation 1.01) and judged positively the usefulness of the tool (mean 3,5, SD 0.9). About 40% of the users reported that the number of discovered errors ranged between 20 and 40% while for another 40% of them, the discovered mistakes ranged from 40 to 60% (see series "Q3" in Fig. 4)[9]. We also noted a correlation between the degree of user expertise and the number of mistakes found by the tool, which was higher for less skilled users.

The students reported good comments on the use of the tool since it helped to solve problems due to unexpected behavior of the presentations without requiring a big effort to learn to use it. On an average, they evaluated their experience to 3, 6 (series "Q4" in Fig. 4), with a standard deviation of 0.7 and a most frequent value equal to 4. Even the relationship between the help given by the tool during the authoring and the effort to learn to use it was considered good (mean 3, 7, SD 0.7, complete distribution depicted by series "Q5"). Moreover, in the additional remarks, some users reported that they especially appreciated the simplicity of use of the tool and gave some suggestions to improve it when the SMIL fragment

[9] The possible answers for series "Q3" are: 0 = 0%, 1 = 0–20%, 2 = 20–40%, 3 = 40–60%, 4 = 60–80%, 5 = 80–100%.

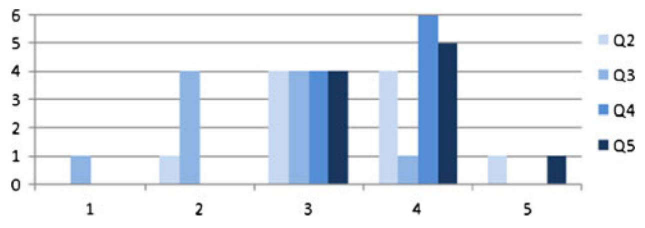


Fig. 4 Answers to question Q2 ("How do you evaluate the usefulness of eSMIL?"), Q3 ("What is the percentage of errors detected by eSMIL?"), Q4 ("Please assess the user experience with eSMIL") and Q5 ("How do you rate the relationship between the help given by the tool during the authoring and the effort to learn to use it?") of the questionnaire

contains events. If the presentation changes its behavior according to user interaction, the tool must know when it occurs, therefore it asks it to the user. The students reported that the tool should be improved in order to suggest this input, otherwise difficult to be calculated.

Summing up, the lessons learned from this experience are

- the use of a validating tool, like eSMIL, allows to eliminate $20-30\%$ of the mistakes usually present in the delivered homeworks;
- many students considered the prototype, beyond an almost complete SMIL validator, also a good tool to improve their knowledge about SMIL since it clearly explains the effect of the definition of elements and attributes and the origin of the mistakes. This means that the tool deeply helps the authoring activity;
- the tool may reduce the time required to create a multimedia presentation and improves the quality of the SMIL code;
- the tool must be user-friendly in order to be useful.

The last point should be carefully taken into account in future developments. As an example, some user reported that the tool can help to discover the wrong definition of events in the attributes `begin` or `end`, e.g., a video ends when a user clicks on a image, but this event cannot occur because that image is not active during the video playback. A future improvement is to calculate the time interval in which each event can occur, and to find out possible mistakes like the one described above. Actually, the user discover this mistake when she/he try to figure out a time instant for that event to input the preconditions. This operation may sometimes require a big effort, that may prevent the user to validate that code.

5 Conclusions

In this paper, we have considered the problem of the complexity of multimedia authoring and we have presented a tool to assist the user to produce SMIL documents with robust and clear code. The tool verifies the presence of temporal conflicts and, in the case of correct documents, produces as output a valid sequence for scheduling. Therefore, it can be used both as a basis for the implementation of a player and as a component of an authoring environment to assist the user to correct mistakes or for the implementation of visualization plug-ins like a timeline view or a preview window.

The tool is based on a formal semantics defining the temporal aspects of SMIL by means of a set of inference rules which describe how the execution of a piece of code changes the state of the computation of a player.

We remark that we do not aim at implementing a new authoring system, but a module which automatically finds temporal conflicts in SMIL documents and helps the user to fix them by providing useful messages. Although many tools have been implemented since SMIL first definition, these facilities are usually still absent. The tool has been tested in an academic environment with positive results.

It is worth noting that all the inference rules can be used both for a top-down construction of a correct playback sequence of the media items involved in the multimedia presentation and for a bottom-up analysis of the SMIL document. This second feature is particularly useful during the context adaption of a document to find out a suitable candidate for substitution or, more in general, during the authoring of the document by composition of elements. Moreover, our rules help in discovering the weakest precondition, i.e., the minimal set of requirements needed to evaluate an element. In our system, this set contains the natural duration of continuous media, the syncbase of the element, which is equal to zero by standard convention for the outer-most element, and information about user interactions.

The soundness and completeness of our approach has been discussed according to an operational semantics which formalizes the changes in the state of a player described informally in the SMIL recommendation. Indeed, the tool passed the test provided by W3C in the SMIL Testsuite [10].

Summing up, the main advantages with respect to the other works addressed in Sect. 2 are the following:

- the definition of a semantics for the standard SMIL 3.0 which covers all the synchronization elements and their most important attributes, while other works consider SMIL 1.0 or a very limited set of elements and attributes of SMIL 2.1 and therefore cannot be of any practical use;
- the proposed semantics allows both the discovery of temporal conflicts and the generation of a valid scheduling for rendering;

Table 13 Proof rule for the parallel composition when the attribute `endsync` is equal to `first`

PAR+ENDSYNC=FIRST

$$\frac{\{A\}\ \texttt{<par c begin='k1' end='stop'}\ \texttt{c}_1 \dots \texttt{c}_n\ \texttt{</par>}\ \{B'\}}{\{A\}\ \texttt{<par c begin='k1' endsync='first'}\ \texttt{c}_1 \dots \texttt{c}_n\ \texttt{</par>}\ \{B\}}$$

where $B = B' \setminus \bigcup_{i \neq j} \{end(\texttt{c}_i) = e_i\}$

APPLICABILITY CONDITION:

$\exists j | ((end(\texttt{c}_j) = min_{\texttt{c}_i}\ \{end_{B_i}(\texttt{c}_i)\}) \wedge stop = end(\texttt{c}_j))$

SMIL SPECIFICATIONS [9]

The endsync attribute controls the implicit duration of time containers, as a function of the children. [...] [If its value is equal to first,] the par, excl, or media element's implicit duration ends with the earliest active end of all the child elements.

- the tool helps the author to fix the mistakes and does not require to change the preferred interface or to know any formalism;
- the users reported that the tool can help improving the user knowledge about the SMIL standard;
- the tool allows a modular evaluation of the elements nested in a SMIL document;
- the compositionality of the approach allows an easy extension of considered SMIL features;
- all the works addressed here require to translate the SMIL document into another formalism, e.g., temporal constraints or a Petri Net, in order to check its temporal consistency. This operation is not always cost-effective, especially when the complexity of the input increases and a non compositional approach is used.

Most of the advantages discussed above derives from the choice of a semantics inspired by the Hoare logic as basis for the formalism. In fact, it allows us to incrementally extend the subset of SMIL features implemented. New features are easily added by defining new rules to describe the semantics of a particular element or attribute, or by defining a translation to a more simple situation, e.g. the CMD+BEGIN+END+DUR translates an element containing all the attributes `begin`, `end` and `dur` into an equivalent element without the attribute `dur`. As a second example, let as consider a parallel container with an attribute `endsync` equal to `first`.[10] The rule PAR+ENDSYNC = FIRST in Table 13 translates the par element to the same composition with a correct value of the attribute `end`, i.e., the earliest time instant in which a child ends. Moreover, our semantics does not need any translation into a new formalism and allows an easily implementation of the rules using the Prolog language as discussed in Sect. 4.1.

Finally, since most available authoring systems adopt "*... SMIL language for building the final representation scheme*" [3], we argue that a formal semantics for this language is needed.

[10] Other values for this attribute are `last`, the default behavior, a media object, that is deprecated since the same behavior con be described with the use of events, and `all`, which is used in the `excl` element.

In the future works, we plan to extend the set of attributes covered by the semantics and to improve the user interface of the tool in order to suggest time interval in which the user interactions must occur. Moreover, the tool must be able to automatically generate sequences of user events to test the SMIL documents.

Currently, the tool verifies only documents for which all information about timing, i.e., the duration of included media, are available. This is not true in all situations, e.g., e-learning where the video of a lesson is streamed in real time, and it could be useful to know, during the lifetime of the presentation, when some time conflicts are generated. It is important to note that we do not need to change the formal semantics to support this possibility, but only the implementation of the tool. One solution can be to check the document more than once, at pre-defined time intervals. The tool will records which media item are finished, and which are still playing. For this second group, the tool can check if the duration accumulated so far exceeds the assigned time interval, e.g., a media is playing longer than its parent element and reports it to the author. A second solution can be the use of parameters, i.e., like in the case of the interval for user interactions, the tool can provide as output an interval for the duration of media items to prevent that their playback will be truncated. This result could be obtained using the *constraint logic programming (CLP)* [12], which extends logic programming to include constraints propagation and solving. CLP allows us to give as result not a single value, but a set of constraints on the expected values. Future work will better investigate this issue.

Acknowledgments The authors would like to thank Mattia Boldrin for his support in the development of the tool and the anonymous referees for the useful suggestions.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)
2. Belkhir, A., Bouyakoub, S.: Formal design of smil documents. In: Proceedings of WEBIST, 2007

3. Bertino, E., Ferrari, E., Perego, A., Santi, D.: A constraint-based approach for the authoring of multi-topic multimedia presentations. In: ICME, pp. 578–581, July 2005
4. Bossi, A., Gaggi, O.: Enriching SMIL with assertions for temporal validation. In: Proceedings of ACM MM, pp. 107–116, September 2007
5. Bouyakoub, S., Belkhir, A.: H-SMIL-Net: a hierarchical Petri Net Model for SMIL documents. In: Proceedings of International Conference on Computer Modeling and Simulation, pp. 106–111 (2008)
6. Bulterman, D.C.A., Hardman, L.: Structured multimedia authoring. ACM Trans. Multimedia Comput. Commun. Appl. **1**(1), 89–109 (2005)
7. Bulterman, D.C.A., Rutledge, L.W.: SMIL 3.0, flexible multimedia for web, mobile devices and daisy talking books, 2nd edn. Springer, Berlin (2009)
8. Bulterman, et al.: Synchronized multimedia integration language (SMIL) 2.1 specification, December 2005
9. Bulterman, et al.: Synchronized multimedia integration language (SMIL) 3.0 candidate recommendation, January 2008
10. Chang, W., Michel, T.: SMIL 2.0 Testsuite
11. Eidenberger, H.: SMIL and SVG in teaching. In: Internet imaging V, vol. 5304, pp. 69–80 (2003)
12. Frhwirth, T., Herold, A., Kchenhoff, V., Le Provost, T., Lim, P., Monfroy, E., Wallace, M.: (1992) Constraint logic programming. In: Comyn, G., Fuchs, N., Ratcliffe, M. (eds) Logic Programming in Action, vol. 636. Lecture Notes in Computer Science. Springer, Berlin, pp. 3–35
13. Hardman, L., Bulterman, D.C.A., van Rossum, G.: The Amsterdam hypermedia model: adding time, structure and context to hypertext. Commun. ACM **37**(2), 50–62 (1994)
14. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–585 (1969)
15. Jourdan, M.: A formal semantics of SMIL: a web standard to describe multimedia documents. Comput. Standards Interfaces **23**(5), 439–455 (2001)
16. Mazouz, S., Dahamani, D., Kaddouri, L.: Formal approach for the coherence control of SMIL documents. Int. J. Comput. Sci. Appl. **3**(2), 126–144 (2006)
17. Schmitz, Patrick, et al.: The SMIL 2.0 Timing and Synchronization Module (2001)
18. Sampaio, P., Santos, C., Courtiat, J.-P.: About the semantic verification of SMIL documents. In: ICME, pp. 1675–1678. New York, USA, August 2000
19. Sampaio, P.N.M., Courtiat, J.-P.: An approach for the automatic generation of RT-LOTOS specifications from SMIL 2.0 documents. J. Braz. Comput. Soc. **9**(3), 39–51 (2004)
20. Valente, P., Sampaio, P.: TLSA Player: a tool for presenting consistent SMIL 2.0 documents. In: Proceedings of ICEIS2007, Madeira, Portugal, June 2007
21. Yang, C.: Detection of the time conflicts for smil-based multimedia presentations. In: Workshop on Computer Networks, Internet, and Multimedia, pp. 57–63 (2000)
22. Yu, H., He, X., Gao, S., Deng, Y.: Modeling and analyzing SMIL documents in SAM. In: MSE, pp. 132–135. Newport Beach, California, December 2002