

Modeling and Analyzing SMIL Documents in SAM

Huiqun Yu, Xudong He, Shu Gao, Yi Deng
School of Computer Science
Florida International University
Miami, FL 33199, USA
{yhq|hex|sgao01|deng}@cs.fiu.edu

Abstract

A composite multimedia object has specific timing relationships among the different types of component media. Coordinating the real-time presentation of information and maintaining the time-ordered relations among the component media is vital to satisfying quality of service (QoS) requirements. This paper proposes a formal approach to modeling and analyzing temporal aspects of SMIL documents using the Software Architecture Model (SAM), which is based on a dual formalism combining Petri nets and temporal logic. Synchronization elements of SMIL are systematically modeled by Petri nets. Useful QoS properties of SMIL documents are specified using temporal logic formulas and verified by automatic tools. Timelines of SMIL document presentation are analyzed by reachability tree technique.

Keywords: Formal method, multimedia, SMIL, SAM, real-time, model, analysis

1 Introduction

Distributed multimedia computing has grown rapidly into a major field of computing research and development. SMIL is the W3C format for multimedia synchronization on the Web [15]. It uses the XML to define a set of markup tags to associate timing and positioning relationships between multimedia objects, such as audio, videos, images and text. SMIL follows an interval-based approach to temporal synchronization. Each media element has an associated presentation interval. These intervals can be coordinated by schedule elements. In general, SMIL defines two kinds of schedule elements. One is a *parallel element* that defines the parallel presentation of n intervals. Using attributes, a more detailed definition of a parallel presentation is possible. For instance, time delays, lip-synchronization, and loops can be specified. The other is a *sequential element* that allows for the sequential presentation of n intervals. Again, a more de-

tailed specification of this presentation is possible with the help of element attributes. The different schedule elements can be nested, thus allowing for the modeling of complex temporal relations.

Most of the works to date addressed the authoring of SMIL documents [9, 16] and the implementation of SMIL players [4, 14], rather than formally defining the precise behavior of the systems to be developed. However, a formal approach can benefit multimedia system design in many ways, such as unambiguous requirement, correct implementation, standardization [2].

This paper proposes a formal approach to modeling and analyzing SMIL documents in SAM. SAM is a general software architecture model for developing and analyzing software architecture specifications [19]. The theoretical basis of SAM is a combination of two complementary formal notions: Petri nets and temporal logic. Synchronization elements of SMIL are systematically modeled by Petri nets. Useful QoS properties such as safety and liveness are specified using temporal logic formulas. The sound mathematical foundation of SAM can help to detect and eliminate design errors early in the development cycle, avoid costly fixes at the implementation stage, and thus reduce overall development cost and increase the quality of the system. Established techniques (such as reachability tree, deductive proof, and structural induction) in SAM are applied to analyzing various real-time properties of SMIL documents.

The rest of the paper is organized as follows: Section 2 gives a brief introduction to SAM and its theoretical basis. Section 3 proposes formal models of synchronization elements of SMIL. Section 4 presents formal techniques for analyzing SMIL documents. Section 5 is the conclusion.

2 The basis of SAM

2.1 SAM models

SAM is a general software architecture model based on two complementary formal notions: Petri nets and tempo-

ral logic. Basically, behaviors of components and connectors are modeled by some kind of Petri nets, while their properties are specified by temporal logic. Formally, a software architecture model in SAM consists of a set of compositions $C = \{C_1, C_2, \dots, C_k\}$ and a hierarchical mapping h relating compositions. Each composition $C_i = \{C_{m_i}, C_{n_i}, C_{S_i}\}$ consists of a set C_{m_i} of components, a set of C_{n_i} of connectors, and a set C_{S_i} of composition constraints. An element $C_{i_j} = (S_{i_j}, B_{i_j})$, either a component or a connector, in a composition C_i has a property specification S_{i_j} (a temporal logic formula) and a behavior model B_{i_j} (a Petri net). Each composition constraint in C_{S_i} is also defined by a temporal logic formula. The interface of a behavior model B_{i_j} consists of a set of places (called *ports*) that is the interaction among relevant components and connectors. Each property specification S_{i_j} only uses the ports as its atomic predicates that are true in a given marking if they contain appropriate tokens. A composition constraint is defined as a property specification. However, it often contains ports belonging to multiple components and/or connectors. A component C_{i_j} can be refined into a lower-level composition C_ℓ , which is defined by $h(C_{i_j}) = C_\ell$.

A SAM architectural specification is *well-defined* if the ports of a component are preserved in the set of exterior ports of its refinement and the proposition symbols used in a property specification are ports of the relevant behavior models. Several correctness criteria of SAM models are identified in [8]. One basic criterion is *element correctness*, that is, the property specification S_{i_j} holds in the corresponding behavior model B_{i_j} , i.e. $B_{i_j} \models S_{i_j}$. Note that we use B_{i_j} here to denote the set of behaviors or execution sequences defined by B_{i_j} .

In this paper, we use predicate transition nets (PrTNs) [7], and a linear-time temporal logic (LTL) [12] as the formal foundations of SAM.

2.2 Behavior modeling using PrTNs

A PrTN is defined with an underlying first-order language \mathcal{L}_P with typed domains. The logic contains ordinary symbols for variables, constants, functions, predicates, logic connectives, and quantifiers. The set of variables V is partitioned into two disjoint sets: D (the set of discrete variables) and C (the set of clock variables). One of the clock variables T is designated as the *master clock*. Let M_s denote multi-set of terms of the special forms $\{k_1 a_1, \dots, k_n a_n\}$, where $k_i \in Nat$ and a_i is a ground term.

A PrTN in \mathcal{L}_P is a tuple $\pi = (P, Tr, F, L, R, m_0)$, where





- P : a finite set of places;
- Tr : a finite set of transitions ;

- $F \subseteq P \times Tr \cup Tr \times P$: the set of arcs;
- $L : F \rightarrow Ms$: a labeling;
- $R : Tr \rightarrow Wff$: a constraining mapping. We assume that R contain both functional constraining mapping R_F and real-time constraining mapping R_T . In particular, a real-time constraint for transition t is a relation of the form $c_t \in [\ell, u]$, where c_t is a local clock associated with transition t , ℓ is called the *earliest firing time* (EFT) for t , and u the *latest firing time* (LFT).
- m_0 : the initial marking.

Remarks: (1) As far as the real-time aspect is concerned, a PrTN is similar to a Time Petri net (TPN) [1]. A real-time constraining mapping R_T in a PrTN is the counterpart of static interval marking (SIM) in TPN. Consequently, they share similar real-time semantics. (2) PrTNs allow for complex data structures. In particular, we require that each token in a PrTN contain a time stamp (global time).

Graphical notations: A real-time constraint for transition t is generally represented by a time interval $[\ell, u]$. For simplicity, we use several graphical notations as defined in Table 1.

Table 1. Several graphical notations

| Description | Instantaneous | Accurate time | Lower-bounded | Upper-bounded |
|-------------|---|---|---|---|
| Notation |  |  |  |  |
| Meaning | $l = u = 0$ | $l = u = a$ | $l = a, u = \infty$ | $l = 0, u = a$ |

A general form for a state S of a PrTN can be defined as a pair $S = (m, I)$ consisting of:

- a marking m ;
- a firing interval set I , which is a vector of possible firing times. The number of entries in this vector is given by the number of transitions enabled under marking m .

Given a marking m , a transition $t \in Tr$ is *enabled* if the following condition holds: $\forall p \in P. (L(p, t) \subseteq m(p)) \wedge R_F(t)$.

A transition t is *firable* from state $S = (m, I)$ at time $\tau + \theta$ if both of the following conditions hold:

- t is enabled under marking m at time τ : $\forall p \in P. (L(p, t) \subseteq m(p)) \wedge R_F(t)$
- the relative firing time θ is not smaller than EFT of t and not greater than the smallest of the LFT's of all the transitions enabled under marking m .

PrTNs use the *strong firing rule*, i.e., an enabled transition t with time interval $[\ell, u]$ under marking m at time τ may not fire before $\tau + \ell$ and must fire before or at $\tau + u$ unless another transition fires before and modifies m . Let us assume transition t be firable at time $c + \theta$ from state $S = (m, I)$. Then the state $S' = (m', I')$ reached from S after firing t at the relative time θ can be computed as follows.

- m' is defined by $m'(p) = m(p) - L(p, t) + L(t, p)$, for all $p \in P$. We call m' the t -successor of m , denoted by mtm' .¹
- I' is computed in three steps:
 - Remove from the expression of I the intervals that are related to the transitions disabled when t is fired.
 - Shift the value θ towards the origin of times for all remaining firing intervals, i.e., the intervals that remain enabled in I , and truncate them, when necessary, to nonnegative values.
 - Introduce in the domain the time interval of newly enabled transitions.

The behavior “transition t is firable from state S at time c and its firing leads to state S' ” is denoted as:

$$S \xrightarrow{(t, c)} S'.$$

A *firing schedule* is a sequence of pairs

$$(t_1, c_1) \bullet (t_2, c_2) \bullet \dots \bullet (t_n, c_n) \dots$$

in which $t_1, t_2, \dots, t_n, \dots$ are transitions and $c_1, c_2, \dots, c_n, \dots$ are times. This firing schedule is *feasible* from a state (m_0, I_0) iff there exist states $(m_1, I_1), (m_2, I_2), \dots, (m_n, I_n), \dots$ such that:

$$(m_0, I_0) \xrightarrow{(t_1, c_1)} (m_1, I_1) \xrightarrow{(t_2, c_2)} (m_2, I_2) \dots \xrightarrow{(t_n, c_n)} (m_n, I_n) \dots$$

A *run* of a PrTN is a sequence of observations $(m_0, c_0), (m_1, c_1), (m_2, c_2), \dots$ in the above firing sequence. We define the *computation* of π , denoted by $Comp(\pi)$, to be the set of all possible runs.

2.3 A linear-time temporal logic

The requirement specification language is a linear-time temporal logic (LTL). A *temporal formula* is constructed from state formulas to which we apply the boolean connectives and temporal operators. Future temporal operators include \square (always), \mathcal{W} (wait-for), \diamond (eventually), \bigcirc (next),

¹Note that multiple non-conflicting transitions are capable of being fired simultaneously. In this case, m' equals to the final marking by arranging transitions to be fired one by one.

and \mathcal{U} (until). Past operators include \ominus (previously), \boxminus (so far), \diamond (once), and \mathcal{S} (since). We write $p \Rightarrow q$ as an abbreviation of $\square(p \rightarrow q)$. A *past formula* is one that contains no future temporal operators.

A *model* for a temporal formula p is an infinite sequence of states $\sigma : s_0, s_1, \dots$, where each state s_j provides an interpretation for the variables mentioned in p . The semantics of a temporal formula p in a given model σ and position j is denoted by $(\sigma, j) \models p$. We define:

- For a state formula p , $(\sigma, j) \models p \Leftrightarrow s_j \models p$
- $(\sigma, j) \models \neg p \Leftrightarrow (\sigma, j) \not\models p$
- $(\sigma, j) \models p \vee q \Leftrightarrow (\sigma, j) \models p$ or $(\sigma, j) \models q$
- $(\sigma, j) \models \square p \Leftrightarrow (\sigma, i) \models p$ for all $i \geq j$
- $(\sigma, j) \models \diamond p \Leftrightarrow (\sigma, i) \models p$ for some $i \geq j$

Two important classes of properties are: *safety* and *response*.

- Safety properties are those that can be expressed by a formula $\square\psi$, for some past formula ψ .
- Response properties are those that can be expressed by a formula $p \Rightarrow \diamond q$, for some past formulas p and q .

Since clocks are explicit variables in PrTNs, temporal formulas may freely refer to them. This includes, in particular, the master clock T , allowing the natural expression of time-dependent properties. For example, $\square(\ell \leq T \wedge T \leq u \rightarrow p)$ states that p holds over the time interval $[\ell, u]$.

One interesting point is that many useful specifications that are response properties in their untimed version correspond to safety properties when a time bound is added [13]. For example, showing that a system reaches and maintains p within time δ can be expressed as the invariance $\square(T \geq \delta \rightarrow p)$.

A model $\sigma : s_0, s_1, \dots$ satisfies a temporal formula if $(\sigma, 0) \models p$. A temporal formula p that is *valid* over a PrTN π specifies a property of π , i.e., states a condition that is satisfied by all computations of π .

2.4 Analysis techniques in SAM

Basically, to ensure the correctness of a SAM model is to show that all the constraints are satisfied by the corresponding behavior models. Several methods have been proposed to analyze SAM models [19, 18, 8, 21], which include:

- *Reachability tree technique*: A reachability tree is an unfolding for a PrTN, which explicitly enumerates all possible markings or states that PrTN generates. The main advantage of reachability tree technique is that the tree can be automatically generated. Once the tree

is generated, various system properties can be analyzed. The main problem is possible space explosion when a PrTN has too many reachable states or even infinite.

- *Deductive proof technique:* The basic idea is to axiomatize PrTNs and then use the obtained axiom system to prove expected properties. The advantage of this technique is that a syntactic approach rather than a semantic approach is used in verification, which eliminates space explosion problem as in the reachability tree technique. The main problems are that the technique is often difficult to automate and its application requires substantial knowledge of first order temporal logic and general knowledge of theorem proving.
- *Structural induction technique:* The basic idea is that given a temporal logic formula, we try to establish a set of simple formulas that implies the original temporal formula. We examine all relevant transition firings in the PrTN and evaluate the simple formulas. If we can establish that the simple formulas are all induction invariants, we can conclude the original given formula. This technique has similar advantages and disadvantages to those of the deductive proof technique. The difference between them are: (1) this technique avoids explicit use of temporal logic and thus easy to use, and (2) this technique is enumerative in terms of the transitions and works forwards while the deductive proof technique is selective in terms of the rules to be applied and works backward.

3 Modeling synchronization elements of SMIL

3.1 Media object elements

The media object elements allow inclusion of media objects into a SMIL presentation. Media objects are included by reference using a URL. There are two types of media objects: *continuous media* (e.g. video, audio) and *discrete media* (e.g. text, image). Each continuous object has an intrinsic duration. In contrast, a discrete object has no intrinsic duration.

Media object types are simply modeled as *places* in PrTNs. Their timing information is maintained with the intrinsic clocks.

3.2 The par element

The children of “par” element can overlap in time. Its timing attributes include

- *begin:* It specifies the time for the explicit beginning of an element. Its value is either a delay-time or a certain event.
- *dur:* It specifies the explicit duration of an element. Its value can be a clock value, or the string “indefinite”.
- *end:* It specifies the time for the explicit end of an element. Its value can contain same type of values as the “begin” attribute.

The attributes “begin” and “end” correspond to either “delay” or “synchronization”, which can be easily modeled as PrTNs. The general form of the `par` element in PrTN is shown in Figure 1.

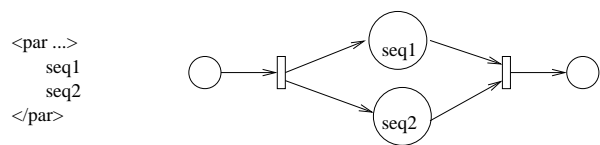


Figure 1. A `par` element and its PrTN

3.3 The seq element

The children of “seq” element form a temporal sequence. It contains the same timing attributes as those of “par”. Figure 2 gives a general form of the element `seq` in PrTN.

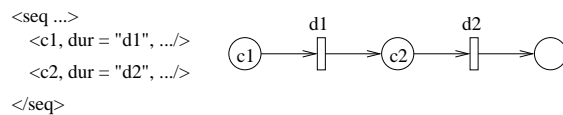


Figure 2. A `seq` element and its PrTN

3.4 The switch element

The switch element allows an author to specify a set of alternative elements from which only one acceptable element should be chosen. Figure 3 gives a general form of the element `switch` in PrTN.

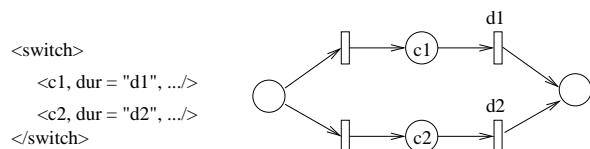


Figure 3. A `switch` element and its PrTN

An example (a SMIL document) Consider the following SMIL document.

```

< id='`par01`' >
  < seq id='`seq01`' >
    < Img1 dur = '5s'.../ >
    < switch >
      < Audio dur = '20s'.../ >
      < Txt dur = '3s'.../ >
    < /switch >
  < /seq >

  < seq id='`seq02`'
    end='`id(seq01)(end)`' >
    < Video dur = '10s'.../ >
    < Img2 .../ >
  < /seq >
< /par >

```

It consists of a sequence of a video clip (*Video*) followed by an image (*Img2*). This sequence (*Seq02*) must be presented simultaneously with another sequence (*Seq01*), which consists of an image (*Img1*) followed by some related information. This information corresponds to an element of the SMIL operator *switch*, such as an audio segment (*Audio*) or a text (*Txt*). The end of presentation of *Img2* is determined by the end of the sequential presentation of *Img1* and the element chosen in the *switch* operator.

Using the above modeling rules, we can obtain a PrTN from the example SMIL document as illustrated in Figure 4.

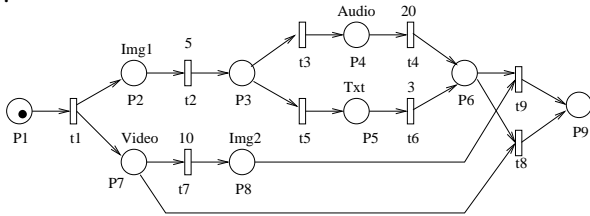


Figure 4. A PrTN for the example SMIL document

4 Formal analysis of SMIL documents

4.1 Synchronization verification

Our verification method is a combination of aforementioned deductive proof technique and structural induction technique, as well as model checking. The method is illustrated in Figure 5, which is based on the Stanford Temporal Prover (STeP) [11]. PrTNs are systematically translated into clocked transition systems (CTSs) [13]. A verification session in STeP begins by loading a program or transition system that describes the system of interest and entering a temporal logic formula that expresses one or more properties to be proved. The formula becomes the root goal of a proof tree. Verification can be performed

by the model checker or by deductive means. The most common route to proceed proof is to apply a verification rule, which generates the first level of subgoals of the proof tree. The basic verification rule for safety properties is B-INV:

$$\frac{\Theta \rightarrow p, \{p\}T\{p\}}{\square p}$$

If all subgoals are established automatically by the Simplifier, the proof is finished. For those verification conditions that are not proved automatically, an interactive Gentzen-style theorem prover is usually invoked.

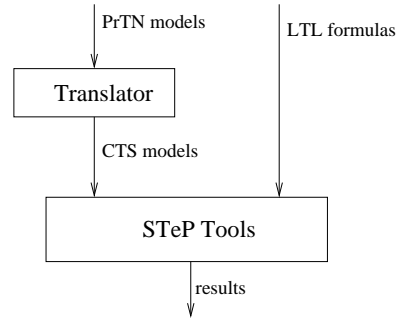


Figure 5. A formal verification method

A CTS $\mathcal{S} = (V, \Theta, \mathcal{T}, \Pi)$ is an extension of transition systems to account for continuous real-time, where V is a finite set of *system variables*, Θ is the *initial condition*, \mathcal{T} is a finite set of *transitions*, and Π is the *time-progress condition*. The reader is referred to [13] for detailed definitions of CTSs.

Given a PrTN $\pi = (P, Tr, F, L, R, m_0)$ as defined in section 2.2, we can define a corresponding CTS $\mathcal{S} = (V, \Theta, \mathcal{T}, \Pi)$, where

- The set of local variables: $V = \{v_p \mid p \in P\} \cup \{c_t \mid t \in Tr\} \cup \{T\}$. Note that each c_t is a *local clock* and T is the *master clock*.
- The initial condition:

$$\Theta = \bigwedge_{p \in P} (v_p = m_0(p)) \bigwedge_{t \in Tr} (c_t = 0) \wedge (T = 0)$$

- For every transition $t \in Tr$, there is a corresponding transition τ_t , whose transition relation is defined as $\rho_{\tau_t} \stackrel{\text{def}}{=} (en(\tau_t) \rightarrow f(\tau_t))$, where

$$en(\tau_t) = R_F(t) \wedge \bigwedge_{p \in P} (v_p \geq L(p, t))$$

$$f(\tau_t) = \bigwedge_{p \in P} (v'_p = v_p - L(p, t) + L(t, p))$$

The set of all possible transitions $\mathcal{T} = \{\tau_t \mid t \in Tr\}$

- The time-progress condition Π is : $\bigwedge_{t \in Tr} (c_t \leq u_t)$

Remark: It has been shown in [20] that both a PrTN and its corresponding CTS have the same observation behaviors.

The example (continued) By applying the translation rules, we can obtain a corresponding CTS $S = (V, \Theta, \mathcal{T}, \Pi)$ from the PrTN in Figure 4, where

- $V = \{v_{p_1}, \dots, v_{p_9}, c_{t_1}, \dots, c_{t_9}, T\}$
- $\Theta = ((v_{p_1} = 1) \bigwedge_{x \neq v_{p_1}} (x = 0))$
- $T = \{t_1, t_2, \dots, t_9\}$
- $\Pi = \bigwedge_{t \in Tr} (c_t \leq u_t)$

The CTS in STeP script is as follows.

```

Clocked Transition System
% A CTS for the SMIL document
out p:array [1..9] of int
where (p[1]=1)
  /\ Forall i:[2..9].(p[i]=0)
clock c1,c2,c3,c4,c5,c6,c7,c8,c9
where c1=0 /\ c2 =0 /\ c3=0
  /\ c4 =0 /\ c5=0 /\ c6=0
  /\ c7=0 /\ c8 =0 /\ c9=0
Progress
(p[2]=1 --> c2 <= 5)
/\ (p[4]=1 --> c4 <= 20)
/\ (p[5]=1 --> c6 <= 3)
/\ (p[7]=1 --> c7 <= 10)
Transition t1: enable p[1]=1
  assign (p[1],p[2],p[7],c2,c7)
  := (0,1,1,0,0)
Transition t2: enable p[2]=1 /\ c2>=5
  assign (p[2],p[3]) := (0,1)
Transition t3: enable p[3]=1
  assign (p[3],p[4],c4) := (0,1,0)
Transition t4: enable p[4]=1 /\ c4>=20
  assign (p[4],p[6]) := (0,1)
Transition t5: enable p[3]=1
  assign (p[3],p[5],c6) := (0,1,0)
Transition t6: enable p[5]=1 /\ c6>=3
  assign (p[5],p[6]) := (0,1)
Transition t7: enable p[7]=1 /\ c7>=10
  assign (p[7],p[8]) := (0,1)
Transition t8: p[6]=1 /\ p[7]=1
  assign (p[6],p[7],p[9]) := (0,0,1)
Transition t9: enable p[6]=1 /\ p[8]=1
  assign (p[6],p[8],p[9]) := (0,0,1)

```

One real-time requirement of the SMIL document is synchronization between the two media sequences, i.e.,

Once the two media sequences start to play, they will coincide within 25 time units .

This property can be formally expressed by the following LTL formula:

$$[] (T \geq 25 \rightarrow p[9]=1).$$

This property is inductive, that is, it is preserved by all transitions. It is proved automatically using rule B-INV and automatic simplification procedures, which is shown in Figure 6.

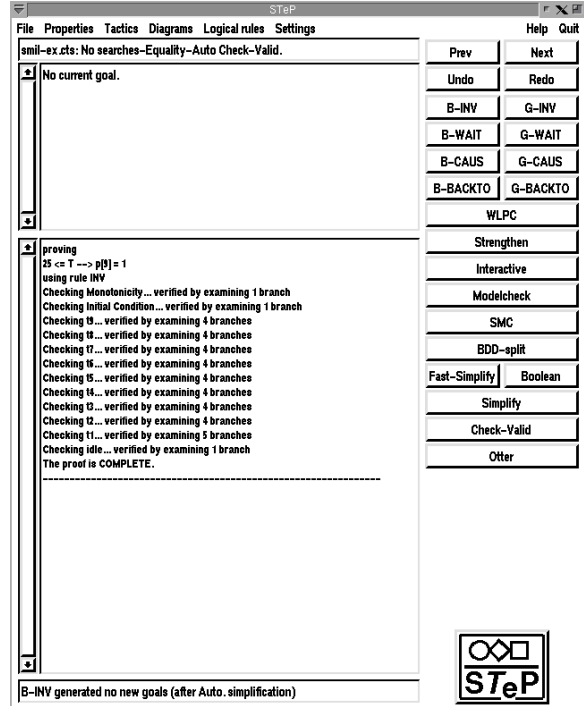


Figure 6. A snapshot of the verification session

Remarks: (1) Important properties (such as safety and liveness) can be expressed as LTL formulas. The STeP-based verification method is suitable for verifying these properties. (2) Certain properties such as *non-Zenoness* and *schedulability*, however, are properties with regard to the existence of some runs, which cannot be expressed within LTL [6]. In these cases, we may resort to several other analysis techniques such as the reachability tree technique.

4.2 Timelines of medium presentation

Since a SMIL document can be modeled by a PrTN, it is possible to automatically derive reference timelines for the SMIL document presentation. Formally, let π be a PrTN resulted from a SMIL document. We can obtain $Comp(\pi)$ via the reachability tree technique. A run is a path going from the root of the reachability tree to a leaf node of the tree. The set of all runs is $Comp(\pi)$. We describe below

how to generate the reachability tree, as well as $Comp(\pi)$, and how to acquire the reference timelines.

Each node in the reachability tree contains state information - the marking and the clock, which is the absolute time calculated from the start. We use a stack to keep the information of alternative firable transitions in a switch condition. Each stack element is a structure that contains a group of firable transitions, together with the marking and the clock. In addition, a string r is used to record the transition firing sequence. A firing sequence is like “(t1)(t2)(t4t7)”, in which transitions in a pair of parentheses are fired simultaneously.

The reachability tree can be generated by the following steps.

1. The root node contains the initial marking and the initial clock, which is 0. Initially, the string r is empty.
2. Collect enabled transitions, and update the firing intervals of each enabled transitions following the rules given in section 2.2.
3. If there is no enabled transitions, then
 - (a) If the string r is not empty, then r contains a run. Add it to the set $Comp(\pi)$. Empty the string r .
 - (b) If the stack is not empty, pop up a group of firable transitions (or maybe just one transition). Restore the corresponding marking and clock, as well as the string r . Fire these transitions at the same time, increase the execution time by their relative firing time. Make the new marking and the new clock as a new node. Put the names of transitions just fired into a pair of parentheses to form a string, and concatenate this string to the string r . Go to Step 2.
 - (c) If the stack is empty, then terminate the whole process. Reachability tree is generated.
4. Select the transition(s) with the earliest relative firing time from the enabled transitions.
5. If more than one transition selected are firable simultaneously, then examine if there is any conflict (switch condition). If conflict exists, then divide the firable transitions into mutual exclusively firable transition groups. Choose any one of these transition groups to be fired. Push the other firable transition groups together with the state information and the current transition firing sequence r into the stack.
6. Fire this transition or these transitions simultaneously. Increase the execution time by its or their relative firing time. Make the new marking and the new execution time as a new node. Put the name(s) of transition(s)

just fired into a pair of parentheses to form a string, concatenate this string to the string r . Go to Step 2.

Once $Comp(\pi)$ is obtained, the reference timelines are easily derived from it. We just use the state information in the nodes of the reachability tree, which consists of the marking and the clock. We then use line chart, with the X-axis representing time and the Y-axis representing the types of media objects. In the same transition firing sequence, if a place contains a token between two clock values, then we draw a horizontal bar to indicate the time duration of the event represented by that place.

The example (continued) Following the above algorithm, the result computation of the PrTN in Figure 4 contains two runs, with one illustrated in Table 2.

Table 2. A run of the example PrTN

| Marking m_i | Clock c_i | Fired transition(s) |
|---------------|-------------|---------------------|
| (100 000 000) | 0 | t_1 |
| (010 000 100) | 0 | t_2 |
| (001 000 100) | 5 | t_5 |
| (000 010 100) | 5 | t_6 |
| (000 001 100) | 8 | t_8 |
| (000 000 001) | 8 | (None) |

The corresponding timeline of medium presentation is shown in Figure 7.

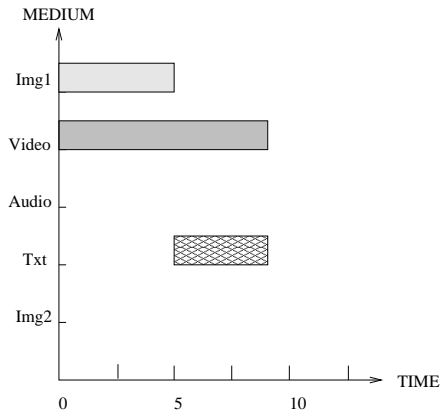


Figure 7. A timeline of medium presentation

A document is considered *consistent* if the action characterizing the *start* of the document presentation is necessarily followed (sometime later) by an action characterizing the *end* of the document presentation [5]. According to this criterion, the example SMIL document is *not* consistent. As we can easily see from the timeline chart, the presentation of *Video* is cut off at time unit 8, instead of specified time unit 10.

5 Conclusion

This paper proposes a formal approach to modeling and analyzing real-time properties of SMIL documents using SAM. Synchronization elements of SMIL are systematically modeled by PrTNs. Useful QoS properties such safety and liveness are specified using LTL formulas. Reachability tree technique is used to compute the reference timelines of medium presentation. Deductive proof, structural induction, as well as model checking techniques are applied to verifying synchronization requirements of SMIL documents. The hierarchical nature of SAM models makes it applicable to complex systems. However, we only used component level model of SAM in this case.

Several other works also addressed formal methods for multimedia computing. OCPN [10] is a well-known multimedia model, which stores its time-control information in places, while its transitions are instantaneous. LOTOS-based reachability graph method was used in [17] to verify temporal properties of IDMs. In [3], symbolic model checking was used for verifying multimedia systems. Our work differs from all of those in several aspects. In contrast to OCPN, transitions in PrTN are not necessary instantaneous. We base our method on the general framework of SAM, which allows for flexible, scalable specification of true concurrent systems. Established techniques, such as reachability tree analysis, deductive proof and structural induction in SAM supported by automatic tools, can be used to efficiently analyze a variety of properties of SMIL documents.

Our future research includes compositional verification techniques for complex SAM models and schedulability analysis of SMIL documents.

Acknowledgements

We thank one anonymous reviewer for the insightful comments, which help improve the presentation of this paper. This work is supported in part by the NSF under grants HDR-9707076 and CCR-0098120, and by NASA under grant NAG 2-1440.

References

- [1] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using Time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.
- [2] G. Blair, L. Blair, H. Bowman, and A. Chetwynd. *Formal Specification of Distributed Multimedia Systems*. UCL Press, 1998.
- [3] S. Campos, B. Ribeiro-Neto, A. Macedo, and L. Bertini. Formal verification and analysis of multimedia systems. In *Proceedings of the 7th ACM International Conference on Multimedia (Part 1)*, pages 419–430, 1999.
- [4] W. Chang and J. Yu. S2M2 - a Java applet-based SMIL player, 1998.
- [5] J. Courtiat and R. Oliveira. Proving temporal consistency in new multimedia synchronization model. In *Proceedings of the 4th ACM International Conference on Multimedia*, pages 141–152, 1996.
- [6] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 16. Elsevier Science Publisher B.V., 1990.
- [7] X. He. A formal definition of hierarchical predicate transition nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets, LNCS 1091*, pages 212–229. Springer-Verlag, 1996.
- [8] X. He and Y. Deng. A framework for developing and analyzing software architecture specifications in SAM. *The Computer Journal*, 45(1):111–128, 2002.
- [9] M. Jourdan, C. Roisin, L. Sabry-Ismail, and L. Tardif. Madeus: An authoring environment interactive multimedia documents. In *Proceedings of ACM Multimedia'98*, pages 267–272, Bristol, U.K., 1998.
- [10] T. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, 1990.
- [11] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. de Alaro, H. Devarajan, H. Sipma, and M. Uribe. STeP: the Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Department of Computer Science, Stanford University, June 1994.
- [12] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
- [13] Z. Manna and A. Pnueli. Clocked transition systems. Technical Report STAN-CS-TR-96-1566, Computer Science Department, Stanford University, 1996.
- [14] RealSystem. *RealSystem G2 Homepage*. <http://www.real.com>.
- [15] W. Recommendation. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*. <http://www.w3.org/TR/REC-smil>, 1998.
- [16] G. Rossum, J. Jansen, K. Mullender, and D. Bulterman. CMIFed: A presentation environment for portable hypermedia documents. In *Proceedings of ACM Multimedia'93*, Anaheim, USA, 1993.
- [17] P. Sampaio, C. Santos, and J. Courtiat. *Using a Formal Method to Verify the Temporal Semantics of SMIL Documents*. <http://citeseer.nj.nec.com/373625.html>, 2000.
- [18] J. Wang, Y. Deng, and G. Xu. Reachability analysis of real-time systems using time Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 30(5):725–736, 2000.
- [19] J. Wang, X. He, and Y. Deng. Introducing software architecture specification and analysis in SAM through an example. *Information and Software Technology*, 41:451–467, 1999.
- [20] H. Yu, X. He, Y. Deng, and L. Mo. Formal analysis of real-time systems with SAM. To appear in *4th International Conference on Formal Engineering Methods*, 2002.
- [21] H. Yu, X. He, Y. Deng, and L. Mo. A formal method for analyzing software architecture models in SAM. To appear in *Proceedings of COMPSAC'02*, 2002.