Modeling Timed User-Interactions in Multimedia Documents

Junehwa Song * *Yurdaer N. Doganata* *Michelle Y. Kim* *Asser N. Tantawi*
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
SONG YURDAER YOON TANTAWI @ WATSON.IBM.COM

Abstract

Interactive multimedia documents (or systems) can be characterized by active user participation and the diversity of multimedia information accessed at various levels of granularity. They need to support extensive user participation in selecting and tailoring the information and its presentation. Multimedia information fragments may vary from portions of video, pieces of audio, newspaper quotations, or chapters of a book. Effectively managing the creation, evolution, and complexity of multimedia documents formed by combining media fragments is an essential capability.

We consider the hyperstory model of a multimedia document, where a document is structured hierarchically in three dimensions: time, space, and asynchrony. The model provides a layered approach in structuring multimedia documents, thus reducing the complexities of large systems. The hyperstory model supports user interactions that are timed, and also supports preemptive resuming. Timed user interactions with the document are modeled with a newly introduced timed Petri-net, TPN*. The TPN* is used to infer the behavior of the system. This paper describes the TPN* modeling, analysis, and application to the hyperstory model.

1 Introduction

With emerging multimedia computing technologies, various forms of interactive multimedia applications are evolving rapidly. As huge amounts of information flood into offices and homes, future interactive systems will be measured in large part by their capability for guiding users through overwhelming volumes of information. In order to give guidance to the users and to make the information more usable, the scope of information needs to be tailored according to the users' needs. As information is made more usable, users will participate more actively; and to make information more usable, users will have to participate more actively. A multimedia system can be viewed as a synthesizer, where information fragments from multiple media sources are combined, or orchestrated, in various ways to convey a meaning. The orchestration occurs in time and space, the result of which is then combined with asynchrony, producing an interactive document. An interactive document thus produced can be viewed as an active entity, in the sense that it can be presented over time and space, while supporting user interactions.

This paper proposes a new formalism, TPN*, for modeling user interactions in multimedia documents (or systems). TPN* is a timed Petri-net (a Petri-net with time) based on Walter's Timed Petri-net[10]. Petri nets have been used for modeling concurrent and asynchronous behavior of systems at various levels[7]. In recent years, its variations with time, timed Petri-nets have been used to model multimedia systems. Their focus has mainly been on capturing synchronization requirements between various media sources. A successful application of a timed Petri-net for modeling asynchronous behavior, caused by user interactions, is yet to emerge. TPN* has been developed specifically to model asynchronous user interactions. TPN* assumes the Hyperstory model[4] as a framework for structuring multimedia documents. The hyperstory model provides a unifying framework for hierarchically organizing multimedia documents. In the model, the static nature of a document, i.e. time, space, and the dependencies between them, is captured in one stratum, and asynchronous, dynamic flow of control (e.g., user-initiated branching) is represented in another stratum above it. This hierarchical structuring helps reduce complexities of large documents.

This paper is organized as follows. Section 2 provides a brief overview of the hyperstory model, and defines *hyperstories*. Section 3 describes how asynchrony in multimedia documents are modeled by the TPN*. It's execution semantics are given, and an algorithm

*The current address of this author: University of Maryland, College Park, MD 20742, JUNESONG @ CS.UMD.EDU

0-8186-7436-9/96 $5.00 © 1996 IEEE
Proceedings of MULTIMEDIA '96

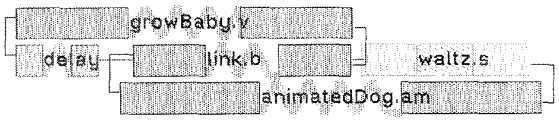

Figure 1: Building a multimedia story using temporal relations

Relation	Objects
co-start	(growBaby, delay)
meet	(delay, link)
co-end	(growBaby, link)
co-start	(link, animatedDog)
meet	(growBaby, waltz)
co-end	(animatedDog, waltz)

Table 2: Temporal relationships among objects

Name	Type	Min	Max	off-set	dur-ation
growBaby	video	15	20	0	15
delay	time delay	2	5	0	2
link	button	10	16	2	13
waltz	sound	10	10	15	10
animatedDog	cartoon	15	25	2	23

Table 1: Type, minimum and maximum durations (in seconds) and a possible schedule for each object

for automatically constructing a TPN* for a hyperstory is provided. An analysis methodology of TPN* is discussed in Section 4. Conclusions appear in Section 5.

2 The Hyperstory Model

In this section, we provide a brief overview of the hyperstory model with an emphasis on its capability for supporting user interactions. For a detailed description of the model, see[4]. The Hyperstory model supports a layered approach to structuring interactive multimedia documents. The model provides a unifying framework for hierarchically organizing multimedia documents. The static nature of a document, i.e. time, space, and the dependencies between them, is captured in one stratum and asynchronous, dynamic flow of control (e.g., user-initiated branching) is represented in another stratum above it. This hierarchical structuring helps reduce the complexity of large documents.

In such a hierarchical structure, an author first describes a portion of a document with a set of temporal relationships treating it as a temporal constraint system. Each object is associated with a minimum duration and a maximum duration over which it can be presented, and a set of four temporal relationships (i.e., *co-begin*, *co-end*, *co-ocurr*, and *meet*) are used to relate the objects temporally. The algorithms for the temporal constraint system compute a run time schedule if the given relationships are consistent. This portion of a document is called a multimedia *story*. See Figure 1 for graphical representation of a multimedia story which we call *lifeStory*. The types and the minimum and maximum lengths over which the objects can be presented are summarized in Table 1. The objects in the story are related as shown in Table 2. The story starts by playing a video (growBaby). After a delay a button (link) appears along with a cartoon (animatedDog). When the video ends, the button disappears at the same time, and a music (waltz) starts. The music and the cartoon will end together. One of the possible run-time schedules produced by the algorithm is shown in the last column of Table 1.

Obtaining a spatial design for a multimedia story is solving an XYT problem, where XY are the two dimensions that define a plane (the display screen) and T is the temporal dimension. The XYT problem is decomposed into a sequence of simpler XY problems, one for each *temporal clique*, by identifying a set of temporal cliques in a story. A temporal clique consists of a set of objects such that each of the objects overlaps with each other at some point in time. As such, objects in a temporal clique can be treated as if they appear on the screen simultaneously. Temporal cliques provide the basis for decomposing a spatial layout of a story into a sequences of screen layouts.

Programs can also be included as part of a document. We assume that a program g is a sequence of executable statements, such that there is one entry point, and that it terminates. A set of programs and a set of stories constitute the static stratum (layer) of a document.

We say that an event is asynchronous if its occurrence and time of occurrence are unpredictable. In the hyperstory model, asynchronous events are defined over multimedia stories and programs. We call a document where a set of stories and programs are connected using asynchronous events a *hyperstory*. We assume that the behavior of a story or a program, when interpreted independently of a hyperstory, is static (predetermined). That is, pressing a button in a story may not result in

an asynchronous behavior unless the story is interpreted in the context of a hyperstory. A hyperstory provides a context for browsing. Asynchrony in multimedia documents is supported through hyperstories.

We now introduce notation to model hyperstories. Let $M = \{m_1, \ldots, m_n\}$ be a set of stories and programs. Then a pair (m_i, m_j), $i, j = 1, \ldots, n, i \neq j$ can be linked by a flow relationship such as branch or call. Flow relationships are conditional; they provide conditional linkages among the stories and programs. Let $U = \{u_1, \ldots, u_k\}$ be a set of user-initiated events. We may assume that u_i is an atomic event (e.g., u_i occurs by the pressing of a button $b_i \in m_i$). Let $D = \{d_1, \ldots, d_n\}$ be a set of events, such that d_i occurs if m_i reaches its end without being interrupted by the user. Note that in our model, each story has associated with it a duration, and thus when the story ends, the end-of-story event is generated automatically. Let $E = U \cup D$. The set $E = \{e_1 \ldots e_l\}$ is the union of the set of user-initiated events, the set of end-of-story and end-of-program events.

We say that m_i is "active" if it is being displayed presently, "on hold" if it has become inactive (i.e., the story is stopped and hidden) but its state is preserved, and "terminated" if it has become inactive without preserving the state. With this, we now give two *flow* relations:

- branch (m_i, e_i, m_j) if e_i occurs while m_i is active, then m_i is terminated and m_j is made active.
- call (m_i, e_i, m_j) if e_i occurs while m_i is active, then m_i is put on hold, m_j is made active; but upon the termination of m_j, m_i becomes active again where it was left off.

We now arrive at a definition of a hyperstory. A hyperstory is a directed graph

$$H = (V, A), \text{ where } V = S \cup G \cup E, \qquad (1)$$

V is the union of three sets, the set S of stories, the set G of programs, and the set E of events, and A is the set of arcs, defined according to the flow relationships among the vertices.

The flow relations allow asynchrony in a hyperstory, in the sense that the flow of control is determined at the time of browsing, rather than at the time of authoring. For instance, while a story is being browsed, a button may appear at a time $t1$ and stay visible until time $t2$. While the button is still visible, it may be pressed causing a branch, or a call. If the story ends without the user touching the button, the end-of-story event occurs and this event may cause a branch according to the flow relationship that has been specified.

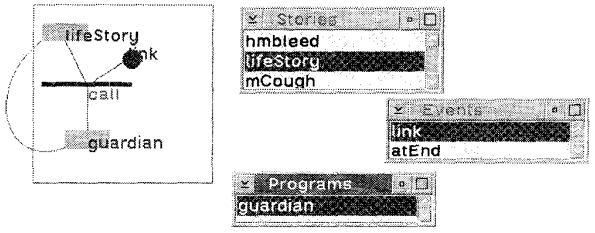

Figure 2: Building a hyperstory

Figure 2 illustrates the building of a hyperstory. A graphical representation of the "call" template is given to the user, where there are three place-holders for source, event, and destination. In the figure, the source has been instantiated by a story "lifeStory", event by a button "link", and destination by a program "guardian". The meaning is that while the story is being played in the context of the hyperstory, if the user touches the button ("link"), the story "lifeStory" is put on hold, and the program "guardian" is called; when "guardian" ends, and "lifeStory" becomes active and starts where it was left off (i.e., it resumes).

3 TPN*: A Petri Net Representation of a Hyperstory

A hyperstory can be represented as a Petri-net. Petri-nets not only capture the descriptive power of directed graphs, but also provide a mathematical abstraction for control and analysis of hypertext "browsing." However, for Petri-nets to be a practical solution for modeling large documents, we must develop a composition technique that produces hierarchically organized documents.

In our hyperstory model, we have separated the static properties, time and space, of a document from the asynchronous, dynamic nature, unpredictable events, providing a two-layered hierarchical structure. Modeling only the (high level) flow relations in the documents as Petri-nets provides us with an opportunity for reducing the complexities. As we shall see shortly, we represent a multimedia story as a *place* in a Petri-net. A multimedia story can also be viewed as a subnet; it is a composite structure. A story provides an abstraction for a set of various static properties in a multimedia document, such as temporal sequencing, and parallelism. Various Petri-net models proposed in the multimedia literature take a flat approach [3, 5, 9], where even simple sequencing operations are modeled by independent places and transitions; they lack a structuring mecha-

nism.

Recently, Petri-nets have been reported in the literature as a tool for modeling concurrent and asynchronous behavior of multimedia systems. As a directed graph, a Petri-net captures the linked structure of relationships among multimedia objects. It is also an automaton, having an execution state and state transition rules, and thus providing browsing semantics for a document.

A timed Petri-net, the object composition Petri-net (OCPN for short), has been proposed in [5] for the formal specification of the intermedia synchronization requirements. The model is based on the logic of temporal intervals [1], and is used to derive a database schema. A Dynamic Time Petri Net model has been proposed in [8], where skip, freeze and restart, reverse, and speed-scaling operations can be supported.

Stotts and Furuta also reported in [9] a timed Petri-net as a formal model of the hypertext [2] The model is based on Merlin's Time Petri Net[6]. In Merlin's Time Petri Net model [6], each transition is labeled by an interval. The interval defines the time duration during which the transition can be fired. Similarly, Walter proposed a Timed Petri Net where a time interval is associated with each incoming arc to a transition[10].

In the rest of this section, we define the TPN*, and describe its execution semantics. We describe the modeling of timed user interactions for decision making and preemptive resumption. Furthermore, we present a short example using TPN*.

3.1 Modeling Timed User Interaction

A multimedia document is a dynamic entity, for it is presented over time, and user interactions with it can also be timed. For example, while a story is being played, a button may appear for a predefined period of time. The user may cause an effect by pressing the button while it is visible on the screen. If the user does not press the button before its time expires, the button may disappear without causing any effect to the flow of browsing the document. We call this style of interaction *timed user interaction*, since a user is allowed to interact within a time window.

The Trellis model [9] supports timed user interactions in multimedia document, where the user can interact with the document during certain time interval. But if there is a time-out, the associated transition fires automatically. In other words, user interactions are merely to speed up the firing of the transition that will be fired automatically if the maximum time is reached without an interaction. In many general cases, users take a more active decision-making role; users choose paths to explore. To support such user interactions, it is necessary to make explicit the consequences of making a choice; consequences of making a choice are different from what will happen if they just let the time expire.

Our objectives for developing TPN* are: (1) to model timed user interactions for *decision-making*, and (2) to support the capability for *preemptive-resumption*. Preemptive-resumption allows a "call" to return at its completion to where it was left off. In our model, there are two classes of transitions: user-initiated and forced transitions. User initiated transitions capture interactive user decisions, which do not assume auto-firing upon time-out. Forced transitions support certain events (e.g., the end of a story) that involve auto-firing.

Definition TPN* is a tuple,
TPN* $= (P, T_F, T_U, F, B, M_0, I, C, R)$ where

- $P = p_1, ..., p_n$ is a finite set of places, $n \geq 0$;
- $T_F = \{t_1, ..., t_o\}$ is a finite set of forced transitions, $o \geq 0$;
- $T_U = \{t_1, ..., t_m\}$ is a finite set of user-initiated transitions, $m \geq 0$, such that $T = T_U \cup T_F, T_U \cap T_F = \emptyset, T \cap P = \emptyset$;
- $F \subseteq (P \times T)$ is the flow relation, a mapping from places to transitions representing (output) arcs;
- $B \subseteq (T \times P)$ is the flow relation, a mapping from transitions to places representing (input) arcs;
- $M_0 = P \rightarrow \{0, 1\}$ is an initial marking for TPN*;
- $I = F \rightarrow R^* \times (R^* \cup \infty)$ where R^* is the set of non-negative real numbers;
- $C = P \rightarrow R^*$ is a mapping from places to clock values;
- $R \subseteq B$ is a set of (input) arcs that resume clock values in the corresponding places;[1]

In TPN*, each place represents a story. A token in a place means that the corresponding story is currently under play. The clock value, C(p), is used to map the state of a place p (i.e., a story). The current clock value represents how much of the story has been played so far from beginning of the story. While a place keeps a token, the clock of the place keeps ticking. The enabling interval, I(a), represents the time window of an event. When a token leaves a place p, its clock c(p) stops ticking, and the state of the place is preserved; As another token enters p via a resuming arc, the clock c(p) continues ticking from where it left off. This is how the preemptive resumption is modeled.

[1] We also call the arcs in $B - R$ resetting arcs.

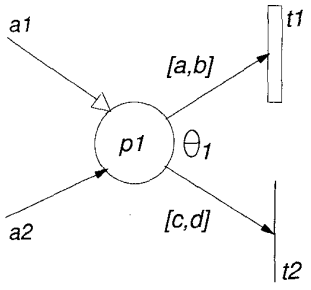

Figure 3: Transitions in TPN*

In our work, we are interested in the user interactions which change the flow of control while browsing documents. Other types of user interactions, such as, fast forward, rewind, pause, etc, can be captured by adjusting the clock behavior.

A graphical illustration of the building block for the model is shown in Figure 3, where p_1 is a place, $t1$ is a user-initiated transition, and $t2$ is a forced transition. Each of the two output arcs from place p_1 to $t1$ and $t2$ has associated with it a time interval [a,b] and [c,d], respectively. The input arc $a2$ is a resuming arc (with an empty arrowhead) which preserves the clock in p_1. The input arc $a1$ is a normal resetting arc (with a filled arrowhead) which resets the clock to its initial value.

3.2 Execution Semantics of TPN*

Each place p has associated with it a local clock $c(p)$. The clock $c(p)$ starts ticking when a token arrives at p and stops ticking when the token leaves p. An output arc $(p_i, t_j) \in F$ is enabled at time τ if at time τ

$$\alpha \leq c(p_i) \leq \beta, \alpha \leq \beta, \qquad (\alpha, \beta) = I(p_i, t_j),$$
$$M(p_i) = 1.$$

We say that an output arc a from a place p_i to a transition t_j is *enabled* if a token is in p_i and the value of $c(p_i)$ is within the range specified by the interval associated with the arc a.

A transition $t_j \in T$ is enabled at time τ if at time τ, (p_i, t_j) is enabled for all i such that $(p_i, t_j) \in F$. That is, a transition t_j is enabled if all the output arcs to it are enabled.

A transition $t_j \in T$ is fired at time τ if

- t_j is enabled at time τ;
- the event associated with t_j occurs at τ.

A user-initiated transition t_j is associated with it an event which is unpredictable (e.g., pressing of a button). Such a transition is fired if the associated event occurs while the transition is enabled. A forced transition t_j is also associated with it an event (e.g., time-out). As in Merlin's model [6], a forced transition must be fired at the moment of time-out if the transition is still enabled. State change caused by transition firing is discussed in the next section.

3.3 Automatic Construction of TPN*

Given a hyperstory, stories in S and programs in G are associated with places, user-initiated events in U are associated with user initiated transitions T_U, and the events in D with forced transitions T_F. The flow relations in A will define the mappings between places and transitions. Timed user interactions are modeled by output arcs (each with an interval) and user-initiated transitions.

Let event e_j denote the event of pressing button j, $start(e_j)$ be the time at which the button j appears, and $end(e_j)$ be the time at which the button j disappears. Intuitively, the event can not occur before the button appears, nor after the button disappears. An event $e_i \in E$ is associated with a time interval $[\alpha, \beta]$, α is the earliest time the event can occur, and β the latest time it can happen. Then,

$$\alpha = start(e_i), \qquad \beta = end(e_i).$$

For instance, in our example summarized in Table 1, the time interval for the button (link) is [2, 15]. As for the end-of-story event in the example, the time interval is [12, 12]. Note that the end-of-story event occurs "at the end" of the story and so the α and β share the same end time point.

We first provide a simpler treatment of a hyperstory, where it does not include a set of programs G. Given a hyperstory H_i with $G = \emptyset$, a TPN* for H_i can be constructed as follows:

- each story $s_i \in S$ is associated with a place $p_i \in P$, such that
  $$f : S \rightarrow P, \quad f(s_i) = p_i.$$
- each event $u_i \in U, d_j \in D$ is associated with a transition $t_j \in T$, such that
  $$g : U \rightarrow T_U \quad g(u_i) = t_j, t_j \in T_U$$
  $$g : D \rightarrow T_F \quad g(d_i) = t_j, t_j \in T_F.$$

We let $\omega_i \in D$ be the end-of-story event for s_i. Then:

- for each story s_i,
  - add an output arc γ to F:
    $$\gamma = (f(s_i), g(\omega_i)),$$
    $$I(\gamma) = [end(s_i), end(s_i)]$$
- for each flow relationship $a_i \in A$,

- if $a_i = branch(s_i, e_j, s_k)$
  1. add an output arc γ to F:
     $\gamma = (f(s_i), g(e_j))$,
     $I(\gamma) = [start(e_j), end(e_j)]$
  2. add an input arc δ to B:
     $\delta = (g(e_j), f(s_k))$
- if $a_i = call(s_i, e_j, s_k)$
  1. add an output arc γ to F:
     $\gamma = (f(s_i), g(e_j))$,
     $I(\gamma) = [start(e_j), end(e_j)]$
  2. add an input arc δ to B:
     $\delta = (g(e_j), f(s_k))$
  3. add an (resuming) input arc θ to R:
     $\theta = (g(\omega_k), f(s_i))$

We add a set of programs G to a multimedia document, such that $G_i \in G$ is a terminating program with single entry point and possibly more than one exit points. Each program G_i is associated with a set Ω_i of end-of-program events, one for each exit point. An end-of-program event, $\omega_{i,j} \in \Omega_i$, is associated with a time time interval $[\alpha_{i,j}, \beta_{i,j}]$, where $\alpha_{i,j}$ is the best case, and $\beta_{i,j}$ is the worst case execution time.

A program G_i can also be mapped to a place $p_i \in P$ in TPN* for a hyperstory:

- each program $G_i \in G$ is associated with a place $p_i \in P$, such that
  $f : G \rightarrow P, f(G_i) = p_i$
- each event $\omega_{i,j} \in D$ is associated with a transition $t_j \in T$, such that:
  $g : D \rightarrow T_F, \qquad g(d_i) = t_j, t_j \in T_F.$

Flow relatinships are treated in a similar way as in the case of the stories.

3.4 Example

Consider a hyperstory as in Figure 2 For the story "lifeStory" as illustrated in Figure 1, a possible schedule is summarized in Table 1. The button "link" appears 2 seconds after the video starts playing, and it remains visible for 13 seconds. If the user touches the button, it calls the program "guardian." Suppose we have another button in the story "lifeStory," and suppose the animatedDog is a button. The the animatedDog button appears at the same time as the button "link" and it remains visible for 23 seconds until the story ends. Furthermore, suppose that when the "animatedDog" button is touched, a story "moreStory" starts.

A Petri-net fragment for the example is shown in Figure 4. Note that the input arc to "lifeStory" is a resuming arc.

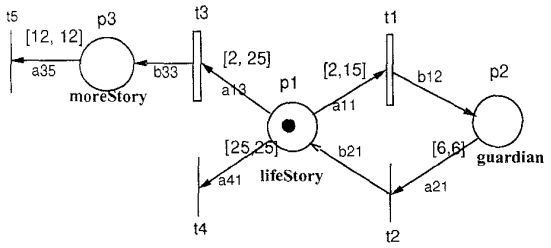

Figure 4: A TPN* fragment for our example

4 Analysis Methodology for TPN*

Once a hyperstory is represented as a Petri-net, many of its interesting properties can be characterized. For instance, the Petri-net can be used to determine if portions of a hyperstory can be reached during browsing, or alternatively, if portions can never be reached.

4.1 States descriptions

A state of a TPN* is defined as a pair $S = (M, \Theta)$. Here, M is the marking, and Θ is a vector of clock values associated with places, i.e., θ_i is the clock associated with the place p_i. When a token arrives at a place, the clock associated with this place is activated and it remains activate until the token is removed. As long as the clock is active, the clock value increases. If there is no token or the clock is not active, then the clock value is the last time that the token departed from the corresponding place. The clock values can be reset through resetting arcs. Hence, clock values can take any real values, $0 \leq \theta \leq \infty$.

Let us consider the representation given in Figure 3. Transition t_1 can fire when $a \leq \theta_1 \leq b$. Transition t_2, on the other hand, can fire when $c \leq \theta_1 \leq d$. Let us assume that $c < a < b < d$. In this case, fireability regions for t_1 and t_2 are as follows:

$$\begin{aligned} 0 \leq \quad \theta_1 \quad < c, \quad & \text{no transition} \\ c \leq \quad \theta_1 \quad \leq a, \quad & \text{only } t_1 \text{ may fire} \\ a \leq \quad \theta_1 \quad \leq b, \quad & t_1 \text{ or } t_2 \text{ may fire} \\ b \leq \quad \theta_1 \quad \leq d, \quad & t_2 \text{ may fire.} \end{aligned}$$

In Figure 5, transition t_2 is enabled by two tokens. In this case, t_2 can fire only if both arcs to t_2 are enabled, i.e., $a \leq \theta_1 \leq b$ and $c \leq \theta_2 \leq d$. If transition t_2 does not fire, then the tokens are removed by the forced transitions t_1 and t_3.

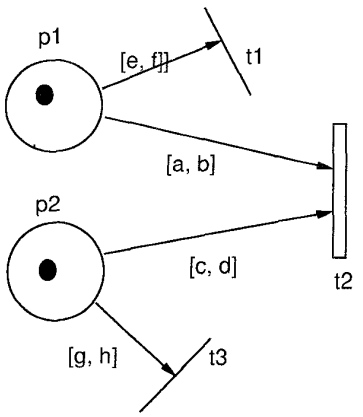

Figure 5: Two tokens enabling a single transition

4.2 State Transitions and the firing rule

Let the state of the system be (M_0, Θ_0) at time z_0. If transition t_1 fires at time z_1, then the next state becomes (M_1, Θ_1) and it is represented as

$$S_0 = (M_0, \Theta_0) \stackrel{(t_1, z_1)}{\Rightarrow} S_1 = (M_1, \Theta_1)$$

If transition t_1 fires at times other than z_1, then the resulting state has the same marking but different clock values. If a transition other than t_1 fires, then the resulting state may have a different marking than M_1. There are infinite number of states with the same marking, since the state clock values are real numbers, $\theta_i \in \Re^*$.

In order to fire a transition, certain conditions have to be satisfied. The fireability conditions are listed below:

1. All the places that are connected to the transition must be active, i. e., must contain a token.

2. All the arcs that are connected to the transition must be enabled. In other words, the clock values that are associated with the active places must be within the limits of the intervals specified by the arcs connecting the places to the transitions. For example, in Figure 3, the arc to t_1 is enabled when $a \leq \theta_1 \leq b$. In figure 5, however, a set of inequalities specify the firing domain of t_2: $a \leq \theta_1 \leq b$, $c \leq \theta_2 \leq d$, where θ_1 and θ_2 are the clocks associated with places p_1 and p_2, respectively. If the user does not take any action within the time limits, then transition t_2 does not fire.

Consider the TPN* in Figure 4. There are three places, p_1, p_2 and p_3, two user initiated transitions, t_1 and t_3, and three forced transitions, t_2, t_4, and t_5. A token in a place signifies that the corresponding story is running. Let us assume that story "lifeStory" is running. The user can initiate a transition between $[2, 15]$ which interrupts lifeStory and starts program "guardian" or the user may initiate another transition between [2, 25] which starts story "moreStory". If the user decides not to take any action, then "lifeStory" is completed when the clock value associated with "lifeStory" reaches 25. Let us assume that at time $\theta_1 = 8$, the user initiates transition t_1. Then, the token is passed to p_2 through a resetting arc which resets the clock value at p_2 to zero, $\theta_2 = 0$. The clock value of p_1 stops at 8 after this transition. "Guardian" keeps running until $\theta_2 = 6$. At time $\theta_2 = 6$, "guardian" ends by a forced transition which sends the token to its original place p_1. Then, the clock starts ticking again from $\theta_1 = 8$. If the user decides to initiate the second transition at time $\theta_1 = 10$, then the token is passed to p_3, clock θ_1 stops at 10, clock θ_3 is reset and "moreStory" starts running. At this point "moreStory" continues till its end and stops at $\theta_3 = 12$.

Let us represent all these events by using the Petri-net notation. The initial state is $S_0 = (M_0, \Theta_0)$, where $M_0 = (1\ 0\ 0)$, $\Theta_0 = (0\ 0\ 0)$. The corresponding state transitions are

$$S_0 \stackrel{(t_1, \theta_1 = 8)}{\Rightarrow} S_1 \stackrel{(t_2, \theta_2 = 6)}{\Rightarrow} S_2 \stackrel{(t_3, \theta_1 = 10)}{\Rightarrow} S_4$$

where
$\tau = 0$, $S_0 = ((1, 0, 0), (0, 0, 0))$,
$\tau = 8$, $S_1 = ((0, 1, 0), (8, 0, 0))$,
$\tau = 14$, $S_2 = ((1, 0, 0), (8, 6, 0))$,
$\tau = 16$, $S_3 = ((0, 0, 1), (10, 6, 0))$,
$\tau = 28$, $S_4 = ((0, 0, 0), (10, 6, 12))$,
and τ is the global time.

4.2.1 Computing the next state after transition t_i

The reachability relation among the states can be determined by the firing rule which is used to compute the next state. The behavior of the TPN* can then be characterized accordingly.

Let us assume that transition t_i fires at time τ from state $S = (M, \Theta(\tau))$. Then the next state $S' = (M', \Theta'(\tau + \delta))$ is reached from S at time $t_i + \delta$ is computed as follows. Here δ is an infinite smal time element and $\Theta(\tau)$ is the vector of clock values at time τ.

- M' is computed for all places p, as:

$$M'(p) = M(p) - |\{(t_i, p) | (t_i, p) \in B\}| + |\{(p, t_i) | (p, t_i) \in F\}| \quad (2)$$

- All the places that keep their tokens remain active after the transition. Hence, the clock values keep increasing.

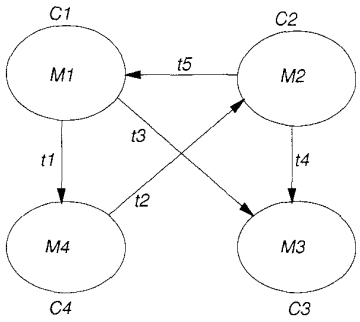

Figure 6: Classes and states

$(\forall p_j)$ such that
$M'(p_j, \tau) = M(p_j, \tau + \delta) = 1,$
$\theta_j(\tau + \delta) = \theta_j(\tau) + \delta.$

- The clock of the place from which the transition takes place becomes inactive.

$$M'(p_j, \tau + \delta) = M(p_j, \tau) - 1 \Longrightarrow$$
$$\theta_j(\tau + \delta) = \theta_j(\tau).$$

- The clock of the place that receives the token is either activated or reset.

$$M'(p_j, \tau + \delta) = M(p_j, \tau) + 1 \Longrightarrow$$
$$\theta_j(\tau + \delta)) = \begin{cases} \theta_j(\tau) + \delta, & \text{via resuming arc} \\ 0, & \text{via resetting arc.} \end{cases}$$

Here, $M(p_j, \tau) = 1$ when there is a token at place p_j. Otherwise, it is equal to zero.

4.3 Definition of State Classes

The states that have the same marking can be grouped to define a state class. The state classes can then be used for modeling and analysis purposes. A state class $C = (M, .)$ is defined as the set of states with the same marking M,

$$C(M, .) = \bigcup S(M, \Theta).$$

Figure 6 shows the relation between states and state classes. In the figure, all the states with the same marking M belong to the same class C_1. These states have different clock values. In general, a class may have infinitely many states, since the clock values are real. The transitions between classes occur as a result of firing a transition. The fireability interval or domain of each transition, however, depends on the initial class. This will be clarified later.

4.3.1 Fireability domain of a state-class transition

When a transition fires, the state of the system changes based on the firing time. As was discussed previously, the next state can be obtained by computing the new clock values and the new marking. As a result, the fireability interval of a transition, which is defined as the time interval during which the transition is capable of firing, changes. The lower bound of the interval is the minimum amount of time that has to be awaited before the transition is fired. Similarly, the upper bound is the maximum waiting time until the transition fires. Here, the time is measured relatively to the time that the token is entered to its new place or that the state has changed.

In Figure 4, arc a_{11} connects place p_1 to transition t_1. Initially transition t_1 cannot fire until arc a_{11} is enabled. The interval associated with arc a_{11} is the *fireability interval* of transition t_1. Therefore, the fireability domain or the fireability interval of t_1 is $[2, 15]$ which is defined as $D(t_1, \Theta) = [2, 15]$, where Θ is the vector of clock values. The interval shows the minimum and the maximum amount of time that a token has to wait until t_1 is fired. After the second transition, however, the fireability interval of t_1 changes. When the token comes back to place p_1 a second time, the clock vector becomes $\Theta = [8\ 6\ 0]$. Since, arc b_{21} is not a regular resetting arc, the value of θ_1 does not change after the token left p_1 for the first time when $\theta_1 = 8$. So, when the token arrives, it finds $\theta_1 = 8$. The value of θ_1 is within the enabling interval of a_{11}, $2 \leq \theta_1 = 8 \leq 15$. Hence, arc a_{11} is enabled immediately. Therefore, the fireability interval for t_1 as a function of the clock values is represented as follows:

$$D(t_1, (0, 0, 0)) = [2, 15],$$
$$D(t_1, (8, 6, 0)) =$$
$$[max\{0, 2 - 8\}, max\{0, 15 - 8\}] = [0, 7]. \quad (3)$$

This means that initially t_1 cannot fire before 2 and after 15 time units. In general however, the fireability domain of t_1 depends on the value of θ_1 which can take any value within the range. The computation of the upper and the lower bounds of this range is an important part of the analysis. The fireability domain that is expressed by equation (3) assumes that the transition occurs at $\theta_1 = 8$. If the exact transition time is not known, then θ_1 is assumed to take any values within $[\ell_1,\ u_1]$, where $\ell_1 = 2$ and $u_1 = 15$. In this case, when the token comes back to p_1, the fireability domain in equation (3) is expressed as follows:
$D(t_1, \Theta) = [max\{0, 2 - u_1\}, 15 - l_1] = [0, 13].$

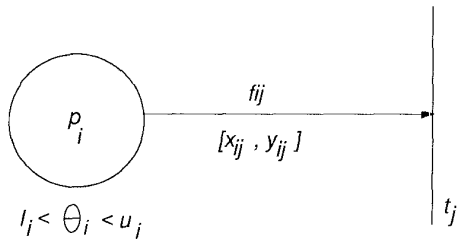

Figure 7: A forward arc between a place and a transition

4.4 Computation of a fireability domain of a transition

For a given TPN*, if the upper and the lower bounds of the clock values are known, then for every transition it is possible to compute the new upper and lower bounds for the clock values. The computation of the fireability domain of a transition involves the calculation of new fireability interval as a result of these changes. If more than one arc is enabled in the system, then the fireability domains of the possible transitions may depend upon each other. As an example in Figure 3. the upper bound of the fireability domain of t_1 depends on d. Since t_2 is a forced transition, it must fire before d. If t_2 fires, then t_1 cannot fire. Hence, the upper bound of the fireability interval is determined by d.

In general, all the fireability domains of the transitions that are enabled by the current marking M are considered in the computation. The upper bounds of the fireability intervals of the forced transitions are considered first, which determine the upper bound of all fireability intervals.

Consider the forward arc $f_{ij} \in PxT$ between place p_i and transition t_j, where the enabling interval of f_{ij}, $I(p_i, t_j)$ is expressed as $[x_{ij}, y_{ij}]$ and the upper and lower bounds on the clock value is given as $[\ell_i, v_i]$. This is illustrated in figure 7. Let the matrix $X \in \Re^{NxM}$ denote the lower bounds of all the enabling intervals in the network, where $[X]_{ij} = x_{ij}$, N is the number of places and M is the number of transitions. Also, let the matrix $Y \in \Re^{NxM}$ denote the upper bounds of all the enabling intervals in the network, where $[Y]_{ij} = y_{ij}$.

Let t_j be the transition for which the fireability domain is computed, T'_f is the set of all forced transitions and T'_u is the set of all user initiated transitions that are enabled by the current marking. Transition $t_j \in T'_u$ cannot fire, if one of the forced transitions $t_f \in T'_f$ fire before it. All the transitions with a non-null fireability interval may fire. If transition t_j fires before other enabled transitions do, then for all incoming arcs f_{ij} to transition t_j, the following set of inequalities have to be satisfied for all places p_i that are marked and connected to transition t_j:

$$x_{ij} \leq \theta_i + \ell_j \leq y_{ij}, \quad \forall f_{ij} \neq \emptyset \tag{4}$$

where $\ell_j > 0$ is the elapsed time between the last state is entered and the time at which transition t_j fire. As a result of these inequalities, if ℓ_j is found to be less than or equal to zero, then this means that transition t_j cannot fire. The upper bound of ℓ_j also depends on the upper bound of the elapsed time of all enabled forced transitions.

$$\ell_j \leq \ell_f, \quad \forall t_f \in T'_f. \tag{5}$$

For every possible transition, the new clock values can be found by using (4) and (5). Here ℓ_j is the length of the fireability interval of transition t_j. The following algorithm summarizes the computation of ℓ_j.

Algorithm : Computation of the Fireability Domain

- Assumption : Each place has at most a token at a time.
- Input : (M, D), t_j, $x_{i,k}, y_{i,k}$,
  - M : vector representing the input marking.
  - D : a set of linear inequalities defining the current clock values.
    Linear inequalities are involving the variables, θ_i's, specifying the clock values of places, p_i's.
  - t_j : the transition of interest.
  - $x_{i,k}, y_{i,k}$: the enabling interval of the arc, $f_{i,k}$.
- Output : (M', D') ,
  - M' : new marking after transition, t_j, is fired.
  - D' : new set of linear inequalities defining the clock values after transition t_j is fired.

1. For each transition, t_k, enabled under marking M, without considering the timing requirement, introduce a new variable ℓ_k, for elapsed time till transition t_k is fired.
   For each $f_{i,k} \in$ F, insert the following inequalities to D',
   $x_{i,k} - \theta_i \leq \ell_k \leq y_{i,k} - \theta_i$,
   $\ell_k \geq 0$.

2. For each forced transition, t_f, insert the following to D'
   $\ell_j - \ell_f \leq 0$

3. For each place, p_i, insert a new relationship defining the new clock value, θ'_i, as follows.

- For each place, p_i, which has a token, i.e. M(i) = 1, $\theta'_i = \theta_i + \ell_j$.
- For each resetting arc, $\gamma_{j,n}, \theta'_n = 0$.
- For other places, $\theta'_i = \theta_i$.

4. Calculate M' from M and t_j by equation 2.

5 Conclusion

We have presented a timed Petri-net TPN* for modeling timed user-interactions in multimedia documents. The TPN* allows users to interact with a multimedia document within a time window, and it supports preemptive resumption. Our analysis methodology makes it possible to analyze a Petri-net with user initiated transitions. The behavior of the Petri-net at a given state and time can be obtained by identifying the transitions that can fire and by calculating their fireability intervals. The method also can be used in the design of a TPN* to avoid deadlocks and to guarantee the reachability of a set of state classes by using the algorithm in the previous section as a tool to accomplish this.

References

[1] J. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11), 1983.

[2] J. Conklin. Hypertext: An introduction and survey. *IEEE computer*, 20(9):17–41, 1987.

[3] M. Diaz and P. Senac. Time stream petri nets: A model for timed multimedia information. In *15th Conf. on Application and Theory of Petri-nets*, 1994.

[4] Michelle Kim and Junehwa Song. Hyperstories: Combining time, space, and asynchrony in multimedia documents. *under revision for ACM Transactions on Information Systems.*

[5] T. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *IEEE Journal on Selected Areas in Communications*, 8(3), 1990.

[6] P. Merlin and D. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Transactions on Communications*, 9, 1976.

[7] J. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, 1981.

[8] B. Prabhakaran and S. V. Raghavan. Synchronization models for multimedia presentation with user participation. In *Proceedings of ACM Multimedia*, 1993.

[9] P. D. Stotts and R. Furuta. Temporal hyperprogramming. *Journal of Visual Languages and Computing*, 1, 1990.

[10] B. Walter. Time petri-nets for modelling and analyzing protocols with real-time characteristics. In *Protocol Specification, Testing, and Verification III*, pages 149–159, 1983.