MODELING WITH ENHANCED PRIORITIZED PRETRI NETS: EP-NETS

Sheng-Uei Guan and Sok-Seng Lim

Department of Electrical & Computer Engineering
The National University of Singapore
10 Kent Ridge Crescent
Singapore 119 260
FAX: (65) 7791103\ (65) 4570706
Email: eleguans@nus.edu.sg\engp9092@nus.edu.sg

ABSTRACT

Recently, many researchers have their attention focused on interactive temporal models, like the extended finite state machines or the extended Petri net models. One of the recent mechanisms proposed for synchronization is known as the Prioritized Petri net, P-Net, which is implemented in the Distributed Object Composition Petri Net, DOCPN. The development of DOCPN has achieved media synchronization in distributed multimedia environments. This P-Net mechanism holds a powerful property: if the deadline is due, it forces firing events regardless whether they are ready or not. A side effect might occur, which is known as premature/late arriving tokens. This paper addresses the key issue of providing flexible multimedia presentation with user interaction and suggests improved P-Net models, which handles premature/late arriving tokens, accommodates dynamic event mechanisms and can specify user interactions in real time during presentation. To demonstrate the concepts are feasible, a prototype with runtime support has been developed and used to construct several interactive multimedia applications, including *skip*, *freeze* and *restart*, *reverse, speed scaling*, and *multicasting*.

KEYWORDS
Interactive multimedia environments, enhanced P-Net, multimedia-authoring, premature/late arriving token, dynamic events, multi-priorities and multimedia synchronization

1. INTRODUCTION

Multimedia systems have become increasingly more sophisticated and complex with the steady improvement of computer technology since the mid-1980s. Trends in computing and communications are changing the way people work, play, and communicate. In today’s world, business demands multimedia applications and communications, which mix audio, video, text and images in real-time, interactive, multiparty communications. Current multimedia applications include videoconferencing, video and audio playback, live video distribution, video libraries and CD-quality sound, multimedia documents, distance learning, shared graphics and images[2].

The focus of the research flows from the synchronization of the multimedia without user interactions, to interactions in distributed environments[2-7]. The meaning of multimedia synchronization and their methods of synchronization are introduced. Multimedia systems operate in two different types of environments: centralized and distributed. In a centralized environment, resources are located locally. Therefore, synchronization in a centralized situation is not a big problem. Where else, in a distributed environment, synchronization could be a disaster if it is not managed properly. The media bases are located at remote areas and, sometimes even, different remote areas for different resources. Although, theoretically, we can provide a tight schedule to maintain the temporal relationships among these data streams, the random network delays still can introduce jitter in each individual stream, and skew among streams at the destination site.

Every researcher has the same goal, which is to improve the Quality of Service (QoS). Multimedia presentations can tolerate deficiencies in presentation quality to a certain degree but no more than that. Human senses cannot perceive slight deviations in the presentation of continuous media e.g. audio and video. This is why multimedia presentations have near real-time requirements. The permitted deviations of the actual presentation quality from the specified presentation quality is captured by means of QoS parameters, such as average delay, speed ratio, utilization, jitter and skew[12]. However, different application domains may have different QoS requirements.

Synchronization issues are divided into two portions; namely inter-media and intra-media synchronization. Inter-media synchronization is to maintain the temporal/spatial relationship between media, e.g. video and audio. Intra-media synchronization is to maintain the temporal relationship within the media, e.g. video usually has 30 frame/sec, and therefore the playback interval is 1/30 sec. Then, inter-media synchronization is further broken down into course-grain and fine-grain. Course-grain refers to the synchronization of presentation stages, and fine-grain refers to the synchronization within a presentation stage, usually between sections e.g. lip-synchronization. However, it must be noted that intra-media synchronization is not the same as fine-grain synchronization.

Inter-media synchronization forms the backbone for intra-media synchronization. They are closely related to each other. At the inter-media synchronization level, it sends a signal to the intra-media synchronization level. Upon receiving the signal, the intra-media level will activate the media to playback. After the playback is completed, it sends another signal back to the inter-media level. This is how the inter-media level oversees the entire performance schedules of each stage or section of the presentation.

Different levels of synchronization have different policies. For intra-media synchronization, we have blocking and restricted blocking policies. A blocking policy could wait for the expected data unit to arrive even when the schedule deadline of the presentation is reached, e.g. static media (text and image) and audio. A restricted blocking policy could discard the late data units when the deadline for presentation is due. For video, late arriving units will be discarded and in the meantime, it replays the data stream for the next stage of presentation. For audio, the late arriving data stream will also be discarded but it will not replay the current unit but hold silent for that stage duration. Then, at the inter-media synchronization level, we have the re-synchronization approaches. There are three types of re-synchronization approaches:

namely parallel first, restricted parallel first and parallel last. These approaches are similar to the master-medium based interactive synchronization discussed in [8].

There are numerous models developed to represent temporal information, and support synchronization in runtime rendering for multimedia systems. Examples are Wahl and Rothernel's Temporal Model[11], Firefly[11] and Fuzzy relation language[11]. Petri nets are designed specifically to model systems with concurrent components. Over the years, scientists have extended the Petri net model to overcome the limitation of its original design such that it can be applied to multimedia systems. Some of the models are Object Composite Petri Net, OCPN[15], Dynamic Timed Petri Net [24], Extended OCPN, XOCPN[17, 22], Transitional Object Composition Petri Net, TOCPN[18], Prioritized Petri Net: P-Net[7], and Distributed Object Composite Petri Net: DOCPN[7]. As we can see, there are so many different extended Petri models with their pros and cons. DOCPN is the most recent extended Petri net model for the interactive distributed multimedia orchestration. It is associated with a powerful property, prioritized arcs, in the net. With that, DOCPN has the ability to carry out the multimedia synchronization, integrated synchronization and interactive synchronization. However, this prioritized Petri net model has a side effect to deal with when transitions are forced to fire, known as the premature/late arriving token (PLAT) problem.

An Enhanced Priority Petri net, EP-net is proposed in this paper. Premature/late arriving token handler is designed to eliminate the side effect. Besides, the EP-net model also shares the same properties as the OCPN model. In other words, the places in the net represent the resources such as video, audio, text and image playback or the interacting activities such as the click of the mouse. And the transitions represent the synchronized points in the presentation. Moreover, the model is associated with the dynamic arcs to simplify the modeling of the interactive multimedia system, including the *skip*, *freeze* and *restart*, *reverse, speed scaling* and *multicasting*.

The paper is organized in the following way. In section 2 we briefly describe the backgrounds and related models for synchronization. Then, we present the EP-net model in section 3. In section 4, the architecture and synchronous control of user interaction is elaborated. In section 5, we explain the authoring environment implemented, corresponding authoring stages and illustration of the EP-net simulator. Finally, section 6 concludes the paper.

2. BACKGROUND AND RELATED WORKS

We have to understand that a multimedia presentation can consist of several media objects with different temporal properties. The playback duration of each object is known as the temporal intervals. There are 13 possible temporal relationships between any two intervals. Figure 1 shows only 7 of the temporal relationships, the remaining 6 are just the inverse. Each block in figure 1 represents the playback duration of a medium, for example video(v) and audio(a). Therefore, user interactions during a presentation might alter the temporal relationship between two related media objects. Table 1[5] depicts the possible changes of the temporal relationships following a user interaction such as the *skip*, *freeze* and *restart*, *reverse* and *speed scaling* operations.

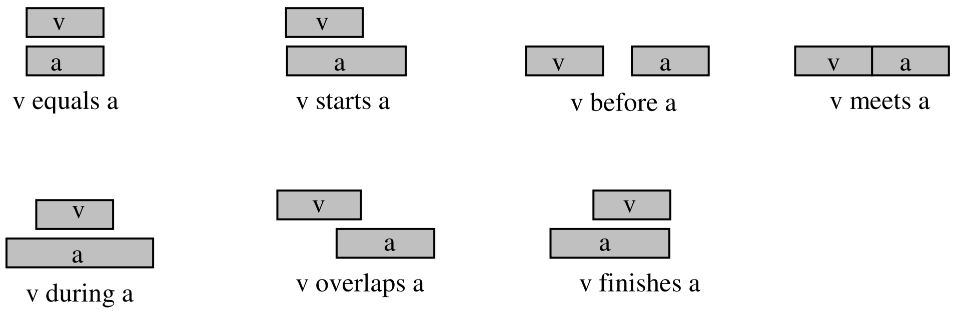


FIGURE 1. Possible temporal relationships between any two intervals[15]

	What it becomes when the user chooses	
Relationship	*Reverse*	*Skip*, *Freeze-restart*, or *scale*
Equal	Equal	Equal
During	No, start, during	No, start, during
Overlap	No, start, overlap	No, overlap, start
Meet	No, meet	No, meet
Before	No, before	No, before
Start	Equal, finish	No, start
Finish	No, start	Finish, equal

Legends:
No: no relationship.
Others: as explained.

TABLE 1. Possible changes of temporal relationships caused by user interactions[5]

2.1 Extensions of Petri Nets

The integrity of a presentation is affected if the temporal constraints are not maintained. Many various models have been proposed to express the presentation specifications, capable of handling unpredictable user interrupts. The most common models proposed are the extended Petri nets. Extended Petri nets have been commonly used to express the dynamics of the multimedia application. In this section, we will briefly discuss the very common extended Petri nets.

2.1.1 Object Composition Petri Net, OCPN

Little and Ghafoor have proposed the use of Object Composition Petri Net, OCPN[15] to model temporal relations between media data in multimedia presentation. The OCPN augments the conventional Petri net model with values of time, as duration, and resource utilization on the places in the net. The OCPN model has a good expressive power for temporal synchronization. However, it lacks of power to deal with user interactions and distributed environments.

2.1.2 Extended Object Composition Petri Net, XOCPN

The Extended Object Composition Petri Net, XOCPN[17, 21] was proposed by Woo, Qazi, and Ghafoor. XOCPN is an upgraded version of OCPN with the ability to model distributed applications. Besides, XOCPN has been made to express fine-grained synchronization requirements for multimedia data. Hence, the temporal interval associated with an object is divided into a sequence of smaller units called synchronization interval units, SIUs. Although XOCPN is able to deal with distributed environments, it cannot handle user interactions.

2.1.3 Dynamic Timed Petri Net, DTPN

The lack of power in OCPN to express user interactions has led to an enhanced OCPN model[16][24], proposed by Prabhakaran and Raghavan. DTPN provides the ability for users to activate operations like *skip*, *reverse*, *freeze*, *restart* and *scaling the speed* of presentation. The dynamic timed Petri net model is suggested to allow user to pre-empt the Petri net execution sequence and modify the time duration associated with the pre-empted Petri net process. The firing rules are similar to OCPN except with a new feature, escape arcs. An escape arc is ended with a dot instead of an arrowhead. A transition t_j with escape arcs may pre-empt the execution if the other normal input places for t_j are active and contain a locked token and at least one of t_j's escape places becomes nonempty. After pre-emption, a locked token is removed from each of t_j's active input place and a token is added to each of the output place of t_j. The four different types[16][24] of interrupts are presented below. An example of *skip* operation is shown in figure 2[16]. Now that this version of OCPN has the power to express interactive multimedia systems, it still faces another problem known as premature/late arriving tokens. The definitions of premature/late arriving tokens are explained in section 2.2.

a. On pre-emption with deference of execution.
b. On pre-emption with termination of execution.
c. On pre-emption with temporary modification.
d. On pre-emption with permanent modification.

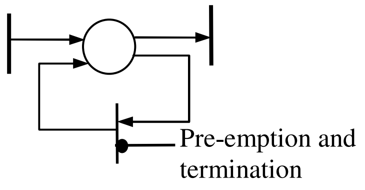


FIGURE 2. *Skip* Operation using enhanced OCPN model[16]

2.1.4 Prioritized Petri Net, P-Net and Distributed Object Composition Petri Net, DOCPN

Guan, et. al have proposed DOCPN[7] to overcome the limitation of the original OCPN and XOCPN in modeling interactive distributed multimedia systems. DOCPN inherits the conventional Petri net firing rule, and applies OCPN and XOCPN synchronous methods to synchronize among inter-media objects. Moreover, it extends

OCPN to a distributed environment using a global clock, and enables user interaction control into the OCPN model. Most important of all, a new mechanism known as prioritized Petri nets(P-nets) are introduced. This priority-input event, as shown in figure 3[7], has the ability to fire a transition without waiting for the arrival of other non-priority input events. More detailed of its firing rules are well explained in [7]. The main concern here is what happens if a token arrives at a non-priority input place, after the transition has been forced to fire by an earlier priority input event. The paper[7] mentions the processing of late arriving tokens depends on the applications; some may choose to discard them (e.g. late arriving video segments), some may choose to recycle or reuse them (e.g. a buffer released for further use). It is better to have a built-in mechanism in P-nets to allow a user/designer to specify how premature/late arriving tokens should be disposed. This motivates the design of EP-nets.

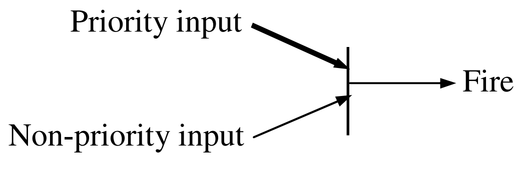


FIGURE 3. A transition with a priority input[7]

2.2 Premature/late Arriving Tokens

Due to the powerful property of priority arcs, it can enable a transition, regardless of the status of other non-priority input places. For an example, in figure 4a, the transition, t_1, has a priority and a non-priority input events, with its respective input places, p_1 and p_2. In the case, when a token arrives at p_1 before the token arrives at p_2, the transition t_1 fires and a token is removed from p_1 and created at p_3. Later, after τ_d time duration after the transition, a token arrives at p_2. Hence, this token is known as a late arriving token. As we can see, the audio and video has gone out of synchronization because of an incoming user interaction (e.g. *skip*), this is undesirable.

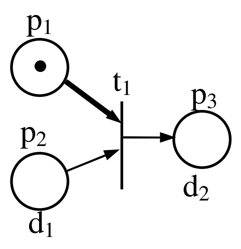


FIGURE 4a. Transition, t_1 fired before any token arrives at p_2

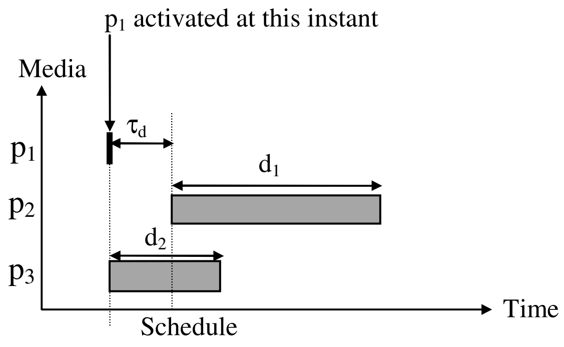


FIGURE 4b. Temporal relationship

Legends:

p_1	: User interaction.
p_2	: Audio clip.
p_3	: Video clip.
d_1	: Duration of audio playback.
d_2	: Duration of video playback.

Although interaction events are unpredictable, it will definitely fall into one of the three situations as mentioned below:

i. When a transition with a priority input event is active, the other non-priority place contains an unlocked token as shown in figures 5a, 5b and 5c.

ii. When a transition with a priority input event is active, the other non-priority place contains a locked token as shown in figures 6a, 6b and 6c.

iii. When a transition with a priority input event is active, the other non-priority place contains no token as shown in figures 7a, 7b and 7c.

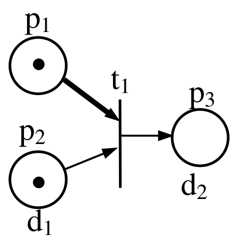


FIGURE 5a. Before the transition

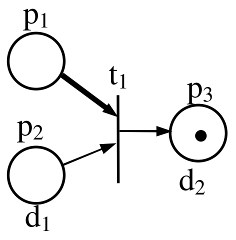


FIGURE 5b. After the transition

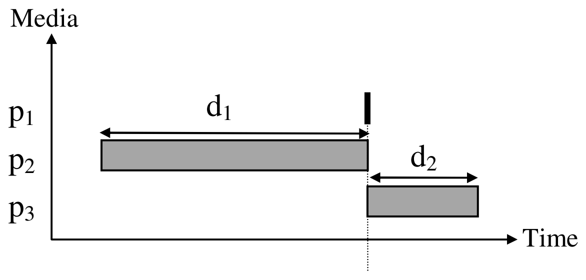


FIGURE 5c. Temporal

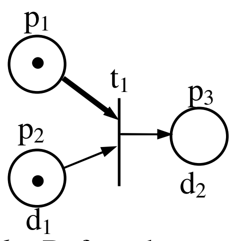


FIGURE 6a. Before the transition

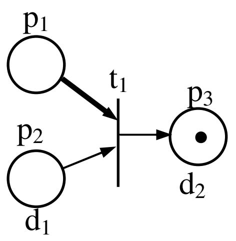


FIGURE 6b. After the transition

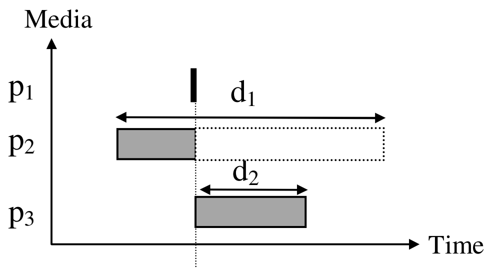


FIGURE 6c. Temporal

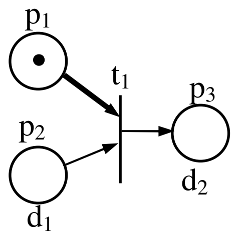


FIGURE 7a. Before the transition

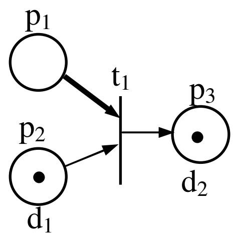


FIGURE 7b. After the transition fires

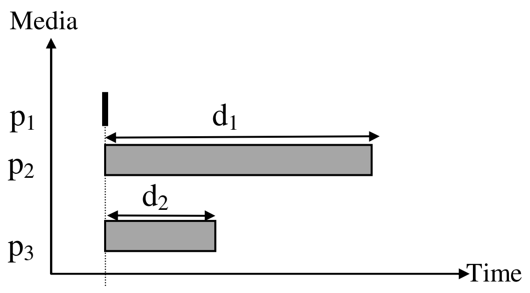


FIGURE 7c. Temporal Information

Figure 5a shows that p_1 (with priority arc) is active when p_2's token is unlocked, this will not create a premature/late arriving token problem. Figure 6a shows that p_1 (with priority arc) is active when p_2's token is locked. Since, p_1 has a higher priority, it forces p_2's token to unlock and enables the transition. Hence, tokens are removed from p_1 and p_2 and created at p_3, and a premature arriving token is forced to fire. This property expresses the EP-net modeling power in interactive environments. Take *freeze* operation for example, if p_1 represents a user interaction and p_2 represents the playing of video clip; the token arrives at p_1 (user interaction occurs) while the token in p_2 is still in the locked condition (e.g. playing of the video clip). The prioritized user interaction forces the playing of the video to stop and keeps the premature arriving token in the premature/late arriving token handler. Figure 7a displays the real problem - late arriving token. p_1 (with priority arc) is active when no token arrives at p_2 yet. However, soon after the transition has fired, a token reaches p_2. This is known as a late arriving token as shown in figure 8. The premature/late arriving token handler (PLATH) shown in figure 8 is explained in section 3.1.

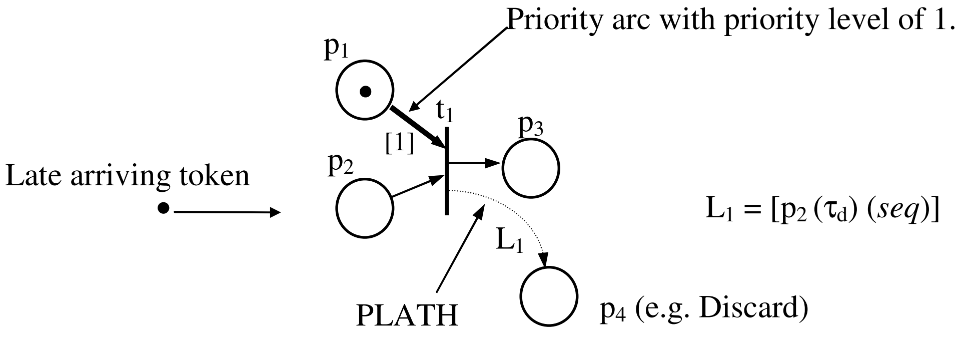


FIGURE 8. Illustration of PLATH, L_1

3. ENHANCED PRIORITIZED PETRI NET

The conceptual temporal model that we are proposing here is an extended model of the Prioritized Petri Nets, P-nets proposed by Guan, Yu and Yang[7]. In this section, an improved version of the P-net, Enhanced P-net, EP-net, is introduced to handle PLAT. Besides, this EP-net imposed another feature that simplifies and improves the flexibility of designing interactive systems, known as dynamic arcs, which can be associated with sets of program statements. Huang has also used this dynamic mechanism, in his extended finite-state machines[5] to orchestrate multimedia presentation, however the main advantage of Petri net over finite state machines is its expressive power to visually display temporal information.

3.1 Definitions

An enhanced prioritized Petri net structure, EP-net, is a twelve-tuple. EP-net = (P, T, I, O, L, D, Pri, DynI, DynO, M, PS, CV). The structure is defined as shown below.

$P = \{p_1, p_2, \ldots\ldots p_n)$ is a finite set of places, where $n \geq 0$.
$T = \{t_1, t_2, \ldots\ldots\ldots t_m\}$ is a finite set of transitions, where $m \geq 0$.
$P \cap T = \varnothing$ i.e. the set of the places and transitions are disjoint.
$\tau = \{\tau_1, \tau_2, \ldots\ldots\ldots \tau_x\}$ is a finite set of remaining duration after the interactions, where $x \geq 0$.
seq = {q} is an integer that indicates the current token sequence number, where $q \geq 0$.
L:T = Bag{P} is the PLAT Handler, PLATH function, a mapping from transitions to bags of places, $L = \{[p_n (\tau_y) (seq)]\}$.
$I:T \rightarrow Bag(P\}$ is the input function, a mapping from transitions to bags of places.
$O:T \rightarrow Bag(P\}$ is the output function, a mapping from transitions to bags of places.
$D = \{d_1, d_2, \ldots\ldots d_w\}$ is a finite bag of duration to bags of places, where $w \geq 0$.
$Pri:(I:T) \rightarrow \{pri_1, pri_2 \ldots.. pri_z\}$ is a priority function, a mapping of priority level to bags of input events, where pri_z is an integer, $pri_z > 0$ and $z \geq 0$.
PS is a set of program statements.
CV is a set of context variables.
$DynI:(P \times Pri \times PS \times CV) \rightarrow Bag\{T\}$ is a dynamic input function, a mapping from places to bags of transitions based on PS and CV.
$DynO:(T \times PS \times CV) \rightarrow Bag\{P\}$ is a dynamic output function, a mapping from transitions to bags of places based on PS and CV.
$M = \{m_0, m_1, \ldots\ldots\ldots m_k\}$ is a finite set of marking, where m_0 represent the initial marking and $k \geq 0$.

The differences of EP-net from the traditional P-net are the introductions of the PLATHs, the multi-priorities and the dynamic arcs. The broken line with an arrowhead as shown in figure 8, represents a PLATH. This PLATH will act as a filter that carry out the post-processing activities when the premature/late arriving tokens are detected. The place, p_4, shown in figure 8 represents any late token detected is discarded from the buffer. The next feature, multi-priorities arc as mentioned in the future work section of [7], is implemented in EP-net. A priority arc is represented by a thick solid line with an arrowhead and associated with its priority level shown in brackets (figure 8). If there is no indication of any priority level, by default it denotes a priority level of 1. So, the higher the priority level indicated, the higher is the priority of the arc. Note that we do not have any priority output event.

As for the last feature, dynamic arcs are associated with a set of program statements. For dynamic arcs, it is most useful when used in applications where flow of events (tokens) can be decided only at run-time. These dynamic arcs can be multi-priorities, non-priority events or PLATHs. The graphical representations of dynamic arcs are shown in figures 9a and 9b. The associated program statements to the dynamic arc as shown in figure 10, deciding where the arc will be pointing to at run-time.

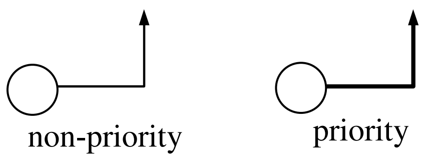


FIGURE 9a. Dynamic input event

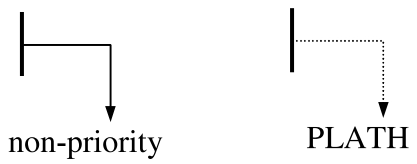


FIGURE 9b. Dynamic output event and PLATH

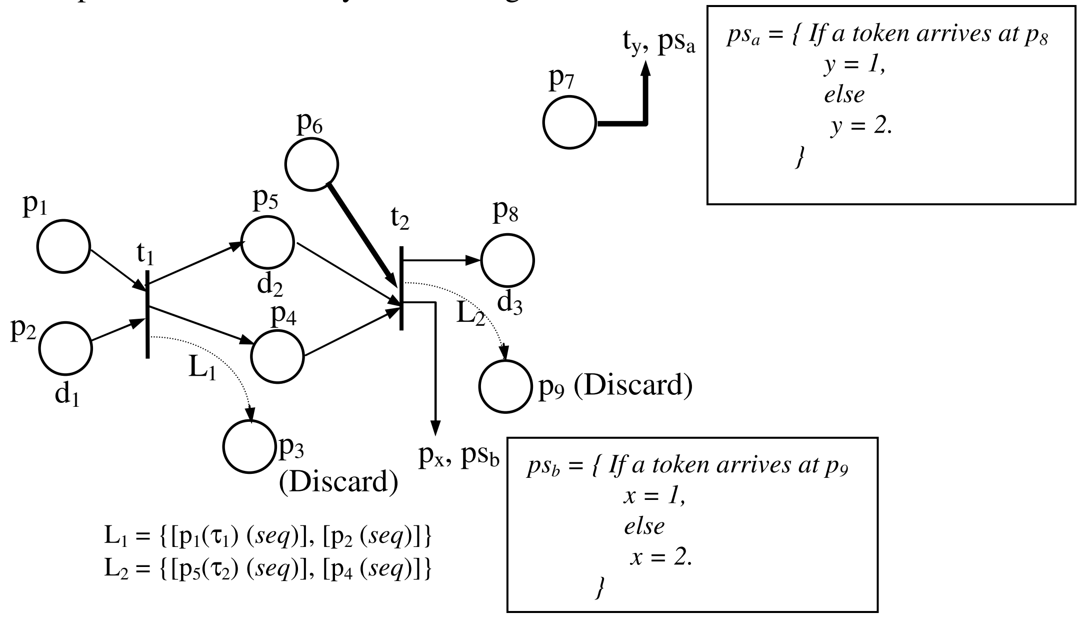
The firing rules of enhanced P-nets, EP-nets are the same as P-nets as mentioned in [7]. A transition with non-priority input events only would fire when all events are complete and ready. A transition with a priority input event concurring could fire with the arrival of that priority input event, without waiting for other non-priority events. For the same priority input events concurring at a transition, we apply the "AND" rule. A place with a token and several transitions enabled from this place will fire the transition with a priority arc from this place. If there are more than one priority arc outgoing from a place enabling more than one transitions, then the firing choice is non-determinate.

The designer has the choice to decide whether it is necessary to include PLATHs. PLATH will deal with any late arriving token reaching a place within a user-specified timeout, τ_d, as shown in figure 4b after the interaction. However, if a premature arriving token still in the locked condition when the interaction occurred, in this case, τ_d, can represent the duration of the media that has been played. Moreover, the designer can also give the choice of detecting the late arriving token based on the duration, τ_d and/or based on the token's sequence number. Each PLATH has a memory space (e.g. register) to store the currently expected token's sequence number, and each token carries its sequence number. Hence, a PLATH matches its stored sequence number with the arriving token's sequence number. If the sequence number does match, this token is considered late. Upon a late arriving token being detected, it will be removed immediate from its respective place and created at the PLATH's place (e.g. P_4 in figure 8). Then, the token is processed as pre-specified for example to be discarded, saved for next use, etc.

3.2 Application and Examples

Based on the example of the global clock controlling the global schedule in [7], including the premature/late arriving token handler completes the DOCPN map. Any premature/late arriving token detected will be discarded, hence the synchronization of the system is maintained and the premature/late arriving token problem is eliminated.

Figure 10 shows an example of the EP-net model. As shown in figure 10, the transition, t_1 will fire if both tokens arrive at places, p_1 and p_2, and the tokens are unlocked. The transition, t_1 will also fire, regardless the states of p_1 and p_2, if the place, p_7, contains at least a token, and the dynamic priority input event points toward t_1. This will force fire t_1 and the premature/late arriving token will be handled by the PLATH L_1. The arrival of token at the place, p_6, will force fire the transition, t_2, and the premature/late arriving token will be handled by the PLATH L_2. The dynamic output arc from p_7 and dynamic output arc from t_2, both depend on the program statements, ps_a and ps_b respectively, to determine which transition or place the arc will be pointing to. For example in ps_b, the dynamic output event pointing to which place depends on whether any late arriving token is detected.



ps_a = { If a token arrives at p_8
$y = 1$,
else
$y = 2$.
}

ps_b = { If a token arrives at p_9
$x = 1$,
else
$x = 2$.
}

$L_1 = \{[p_1(\tau_1)\ (seq)],\ [p_2\ (seq)]\}$
$L_2 = \{[p_5(\tau_2)\ (seq)],\ [p_4\ (seq)]\}$

FIGURE 10. An Enhanced Petri Net

The mathematical representation for the example in figure 10 is shown below:

$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$,
$T = \{t_1, t_2\}$,
$I(t_1) = \{p_1, p_2\}$, $I(t_2) = \{p_4, p_5, p_6\}$,
$O(t_1) = \{p_4, p_5\}$, $O(t_2) = \{p_8\}$,
$L(t_1) = \{p_3\}$, $L(t_2) = \{p_9\}$,
$D = \{0, d_1, 0, 0, d_2, 0, 0, d_3, 0\}$,
$CV = \{x, y\}$,
$PS = \{ps_a, ps_b\}$
$Pri(I(t_1)) = \{0, 0\}$, $Pri(I(t_2)) = \{0, 0, 1\}$,
$DynI(p_7, 1, ps_a, y) = \{t_y\}$,
$DynO(t_2, ps_b, x) = \{t_x\}$.

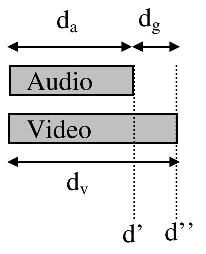
4. USER INTERACTIONS MODELING USING EP-NETS

This proposed EP-Net could be effectively used in synchronization models for flexible multimedia presentation with user interaction. The common user interactions, *reverse*, *skip*, *freeze* and *restart*, *speed scaling* and *multicasting*, are discussed in details in this section. They are modeled using the EP-Net.

4.1 *REVERSE* OPERATION

Reverse operation is one of the common user interactions. When the user requests for a *reverse* operation, the temporal relationship of the multimedia presentation changes as shown in table 1. For an example shown in figure 11a, the "starts" temporal relationships of the media becomes the "equals" or "finishes" temporal relationships after the user requests for a *reverse* operation. We will demonstrate how the *reverse* operation can be modeled by the EP-net.

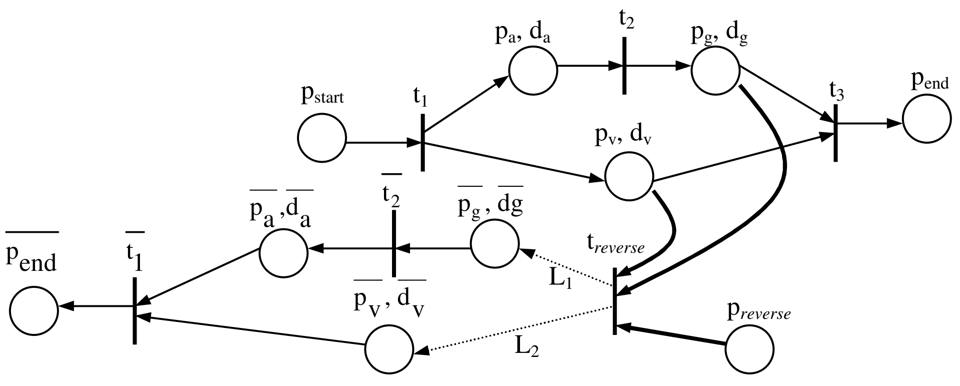
Referring to figure 11a, the multimedia is presented in its normal forward style while the user activates a *reverse* operation during time between d' and d''. The temporal relationship of the media hence becomes a "finishes" temporal relationship. We have designed an EP-net model as shown in figure 11b. According to the EP-net rules, a place with a token and several transitions enabled from this place will fire the transition with a priority arc from this place. The arrival of token at place, $p_{reverse}$, and locked/unlocked token in places, p_g and p_v, will force fire the transition, $t_{reverse}$ and *reverse* the direction of the presentation. The tokens are unlocked and removed, due to its prioritized input event. Since there are no output events from the transition, $t_{reverse}$, no token is created. Then, the PLATH, L_1 and L_2, which deal with the premature tokens of gap and video respectively, will receive the premature token and *reverse* the presentation from the points when they are interrupted. Notice that the *reverse* interaction can only occur when there are tokens in $P_{reverse}$, P_g and P_v, due to the "AND" rule being applied to the same priority level input events concurring at $t_{reverse}$, as shown in figure 11b.



If a user interacts during any time < d', it will become equals relationship.

If a user interacts any time between d' to d'', it will become finishes relationship.

FIGURE 11a. What temporal relationship will become after user interaction

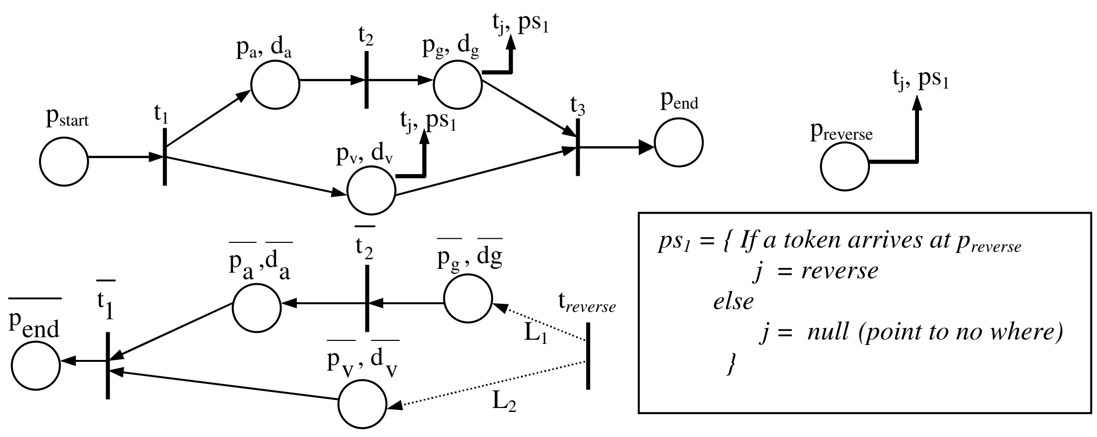


Legends:

p_a	: Place for forward audio playback.
$\overline{p}_a$	: Place for *reverse* audio playback.
p_v	: Place forward video playback.
$\overline{p}_v$	: Place for *reverse* video playback.
p_g	: Place for time lag.
$p_{reverse}$	: Place for *reverse* interaction.
d_a	: Duration for audio playback.
$\overline{d}_a$	: The remaining duration of the premature audio token from p_a.
d_v	: Duration for video playback.
$\overline{d}_v$	: The remaining duration of the premature video token from p_v.
d_g	: Duration for time lag.
$\overline{d}_g$	: The remaining duration of the premature gap token from p_g.
L_1	: Handles (*reverses*) premature arriving token from p_g, $L_1 = \{[p_g(\overline{d}_g)]\}$.
L_2	: Handles (*reverses*) premature arriving token from p_v, $L_2 = \{[p_v(\overline{d}_v)]\}$.
t_1, t_2, t_3, $\overline{t}_1$, $\overline{t}_2$ and $t_{reverse}$	: Transitions.

FIGURE 11b. *Reverse* operation occurs any time between d’ and d’’ (as indicated in figure 11a)

Figure 11c shows how dynamic arcs associated with the program statements can help us simplify the model. Let's us imagine if it is used in an EP-net model which is much larger in size (e.g. increased number of places, transitions and arcs), the crossing of arcs among each other can be eliminated with this dynamic mechanism. For this example we can do so with the statements as in ps_1 (see figure 11c).



```
ps1 = { If a token arrives at p_reverse
            j  = reverse
        else
            j =  null (point to no where)
        }
```

FIGURE 11c. Dynamic arcs associated with program statements for the *reverse* operation

4.2 *SKIP* OPERATION

In the same temporal relationship illustration as shown in figure 11a, consider the skip operation happens during time between d' and d''. The user requests a skip operation to the place P_{end}. An EP-net model is demonstrated in figure 12a.

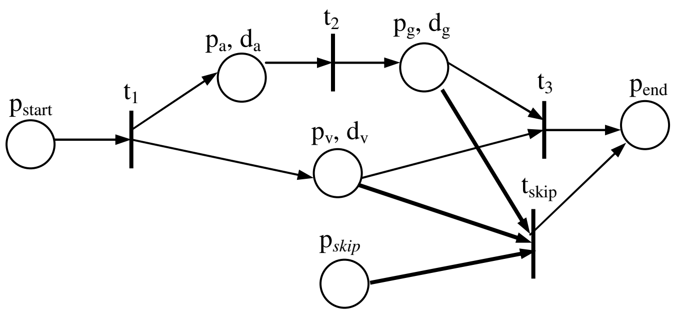
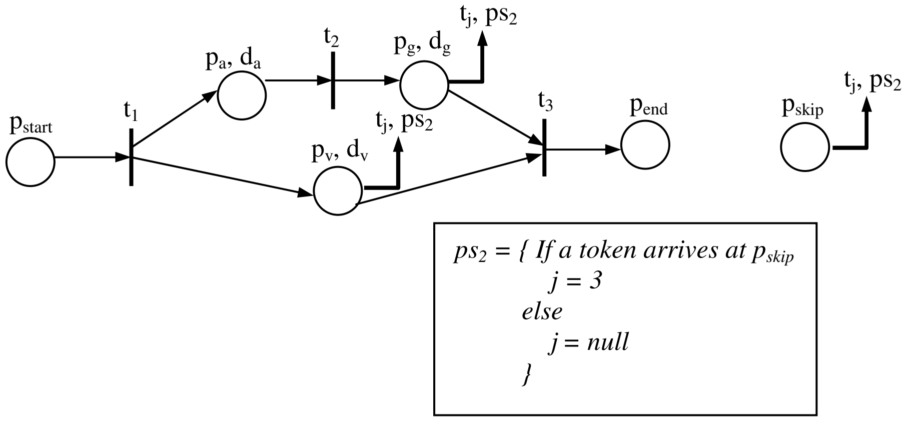


FIGURE 12a. *Skip* operation occurs any time between d' and d'' (as indicated in figure 11a)



```
ps2 = { If a token arrives at p_skip
          j = 3
        else
          j = null
        }
```

FIGURE 12b. Dynamic arcs associated with program statements for the *skip* operation

The legends are similar to those shown in figure 11b except:

p_{skip}	: Place for *skip* interaction.
t_{skip}	: Transitions.
seq	: Sequence number of the tokens.

The *skip* operation using a dynamic mechanism as shown in figure 12b, simplifies the model. It helps to eliminate the needs of the transition, t_4. Since those dynamic input events from p_{skip}, p_g and p_v are not always pointing to transition, t_3, it can function as an interrupt when a token arrives at p_{skip}, regardless whether tokens in p_g and p_v are locked or unlocked. The prioritized arcs from these places stop their playback, which means that they remove the tokens from their respective places, even if the tokens are in the locked condition.

4.3 *FREEZE* and *RESTART* OPERATIONS

This is a dual operation, the user executes a *freeze* operation to stop the presentation, and resumes the presentation by activating the *restart* operation. Using the same temporal relationship as shown in figure 11a, assume the user requests a *freeze* and *restart* operations during the time between d' and d''. Figure 13a demonstrates the *freeze* and *restart* operations at that instance. Moreover, the dynamic version of figure 13a is demonstrated in figure 13b.

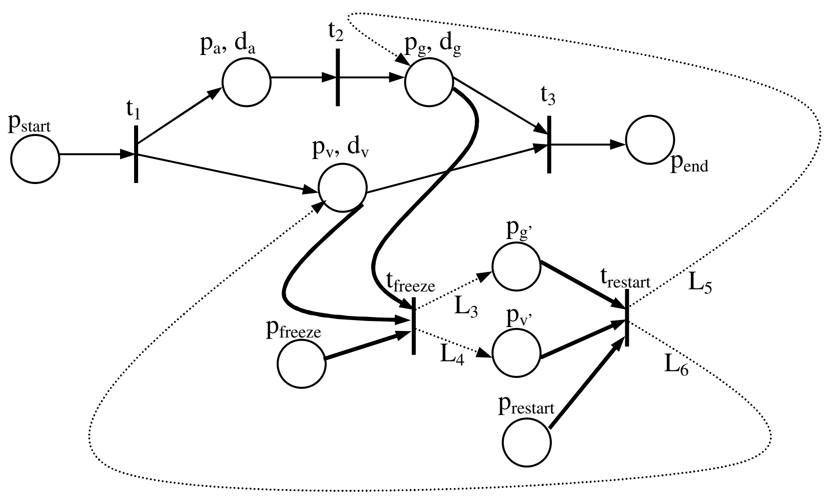
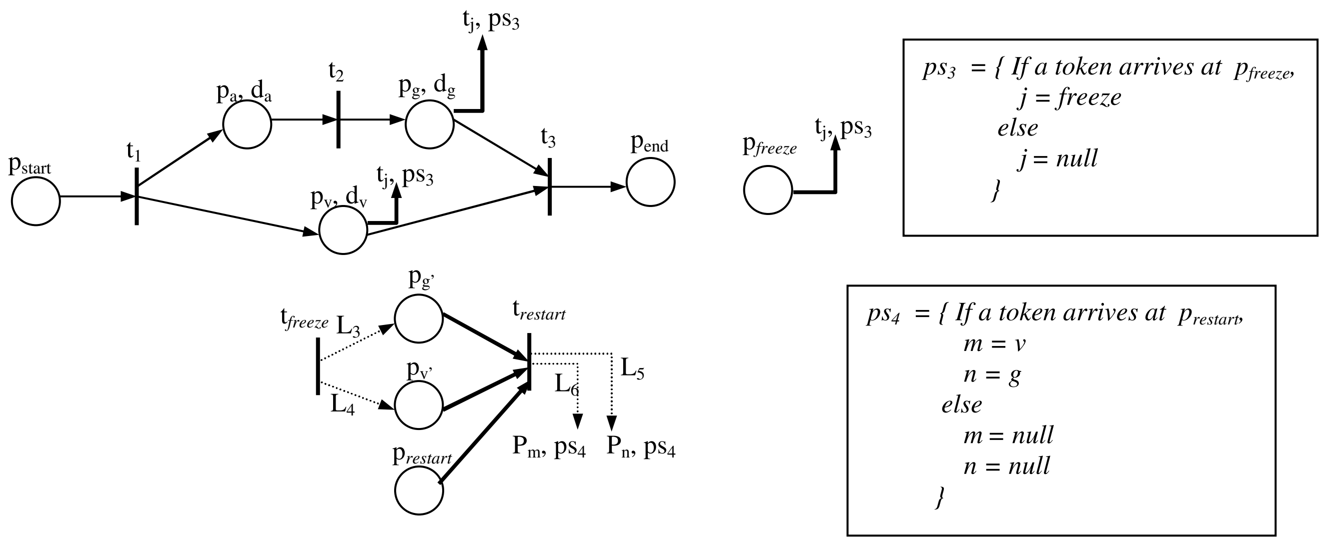


FIGURE 13a. *Freeze* and *restart* operations occur any time between d' and d'' (as indicated in figure 11a)

When the *freeze* operation is activated, the premature tokens in both places, p_g and p_v, is handled by the LATHs, L_3 and L_4 respectively. Then, the places, p_g' and p_v' (e.g. registers) store their corresponding remaining durations. The presentation is resumed when the user activates the *restart* operation.



```
ps3 = { If a token arrives at p_freeze,
            j = freeze
        else
            j = null
        }
```

```
ps4 = { If a token arrives at p_restart,
            m = v
            n = g
        else
            m = null
            n = null
        }
```

FIGURE 13b. Dynamic arcs associated with program statements for the *freeze* and *restart* operations

The legends are similar to those shown in figure 11b except:

p_{freeze}	: Place for *freeze* operation.
$p_{restart}$	: Place for *restart* operation.
p_g'	: Place that stores the duration of time lag that has been played.
p_v'	: Place that stores the duration of video that has been played.
$d_{g'}$	: Duration of the time lag that has been played.
d_v'	: Duration of the video that has been played.
t_{freeze} and $t_{restart}$	: Transitions.
L_3	: Handles (*freezes*) premature arriving token from p_g, $L_3 = \{[p_g(d_g')]\}$.
L_4	: Handles (*freezes*) premature arriving token from p_v, $L_4 = \{[p_v(d_v')]\}$.
L_5	: Handles (*restarts*) premature arriving token from p_g, $L_5 = \{[p_g'(d_g')]\}$.
L_6	: Handles (*restarts*) premature arriving token from p_v, $L_6 = \{[p_v'(d_v')]\}$.

4.4 *SPEED SCALING* OPERATION

Scaling the speed of the presentation can be modeled as illustrated in figure 14a, based on the temporal relationship shown in figure 11a. Assume that the user requests to change the speed of the presentation at time between d' and d''. Figure 14b shows the *speed scaling* operation using dynamic event mechanism.

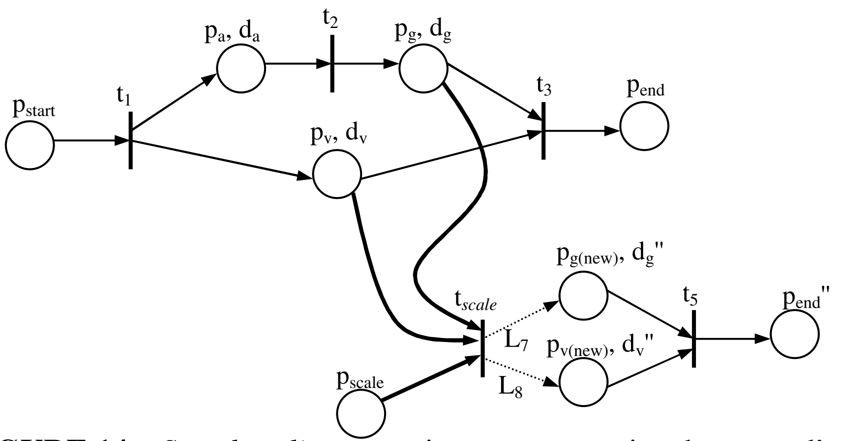
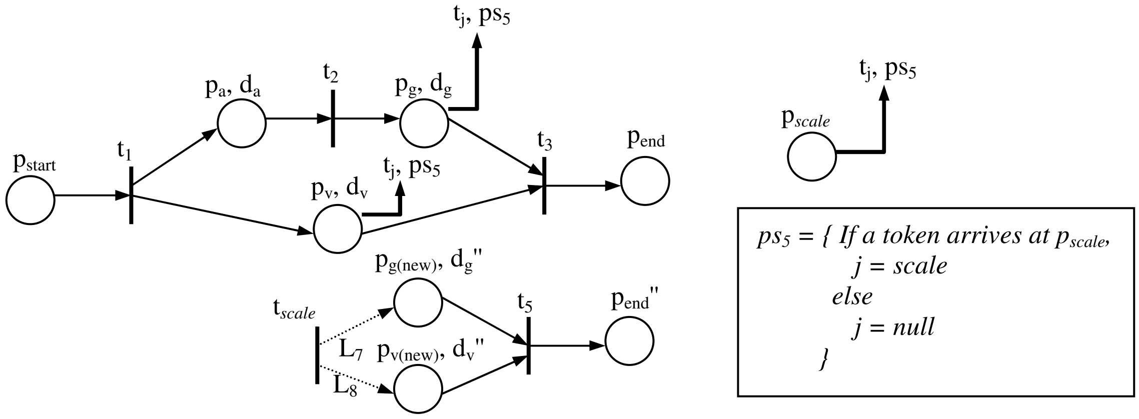


FIGURE 14a. *Speed scaling* operation occurs any time between d' and d'' (as indicated in figure 11a)



```
ps5 = { If a token arrives at p_scale,
        j = scale
      else
        j = null
      }
```

FIGURE 14b. Dynamic arcs associated with program statements for the *speed scaling* operation

The legends are similar to those shown in figure 11b except:

p_{scale}	: Place for *speed scaling* operation
$p_{g(new)}$	: Place for time lag with the new speed, for example ×2, ×4 and etc.
$p_{v(new)}$	: Place for video with the new speed, for example ×2, ×4 and etc.
d_g''	: Duration of the time lag with the new speed.
d_v''	: Duration of the video with the new speed.
t_{scale} and t_5	: Transitions.
L_7	: Handles (*Scale*) the premature arriving token from p_g, $L_7 = \{[p_g(d_g'')]\}$.
L_8	: Handles (*Scale*) the premature arriving token from p_v, $L_8 = \{[p_v(d_v'')]\}$.

4.5 *Multicasting*

The proposed EP-net model when applied to multicasting applications (e.g. VoD multicast) can be implemented based on a multi-threaded server/clients solution. The server is preferably situated on a multi-processor platform to ensure quick response time to service each multicasting client. If tight synchronization is required, priority arcs can be used to enforce tight deadlines when delivering/presenting media contents to each client. Late or premature arriving resources (tokens) on each client will no longer be a problem with the PLATH mechanism. PLATH can simply drop or put away such resources (e.g. for later viewing or use). In the case when flexibility is offered to multicasting clients to change viewing conditions (e.g. scale up or down resolutions), dynamic arcs associated with program statements can come to rescue to change the arrangement of resources and implement corresponding schedule update on-the-fly.

5. AUTHORING ENVIRONMENT

With a suitable online mapping from EP-net to a runtime environment, user interactions modeled in EP-net can be easily simulated. Hence, using the EP-net model, a multimedia synchronization authoring and execution environment can be generated. The authoring and execution environment contains three phases, namely:

Selection and transformation of the media objects to participate: in this phase the author selects the media objects to participate in the authoring environment, and defines the temporal relation of the media objects.

- User interaction: in this phase the author designs the input, output and priority input events, dynamic events and PLATHs.
- Execution: in this phase the designed EP-net model is executed.

We are in the process of developing an experimental EP-net based multimedia synchronization authoring and execution system on Windows (NT) workstations. Users can author the presentation schedule of the system for both centralized and distributed environments. In the near future, we also plan EP-net to be built as run-time library functions so that user applications requiring synchronization can include the required library functions to realize distributed multimedia synchronization.

5.1 EP-NET SIMULATOR

The interrupt specification and handling mechanisms like priority events, PLATH and dynamic events are introduced to enable programmers to specify user interactions. The presentation specification mechanisms like places, non-priority events and transitions are developed to enable programmers to specify temporal relationships among media in a presentation. Together, the distributed interactive multimedia applications can be simulated using this EP-net simulator.

EP-net simulator is designed to be user-friendly. What a programmer needs is a mouse that does most of the job. To draw a place or transition, the user just clicks on the place or transition icons shown on top of the menu as displayed in figure 15, and keys in an integer label from 1 to 50. In the current prototype, we have set its maximum label to 50. Then, by clicking onto any area within the white screen (see figure 15), a place or transition will be drawn. With that, the places and transitions can be linked together with those event mechanisms by clicking respective icons also shown on top of the menu. After the marking is initialized, the simulator is ready to run. This simulator has two running modes. The first mode runs step by step, which means it fires all enabled transitions once and waits for the next execution. The second mode runs and fires till no transition is enabled. The simulator simulates an EP-net as demonstrated in figure 15.

Each place has a local timer. The timer is initialized to a duration value when the presentation starts. The runtime executive in the simulator periodically updates the timer value associated with each active place. For example, places, p_2, p_3 and p_4 are associated with duration, 10, 20 and 10 seconds respectively as indicated in figure 15. If any of these places contains a token (locked), and the user runs the simulator, the duration will count down until zero. Hence, the token in the place is unlocked and ready to be removed if its transition fires.

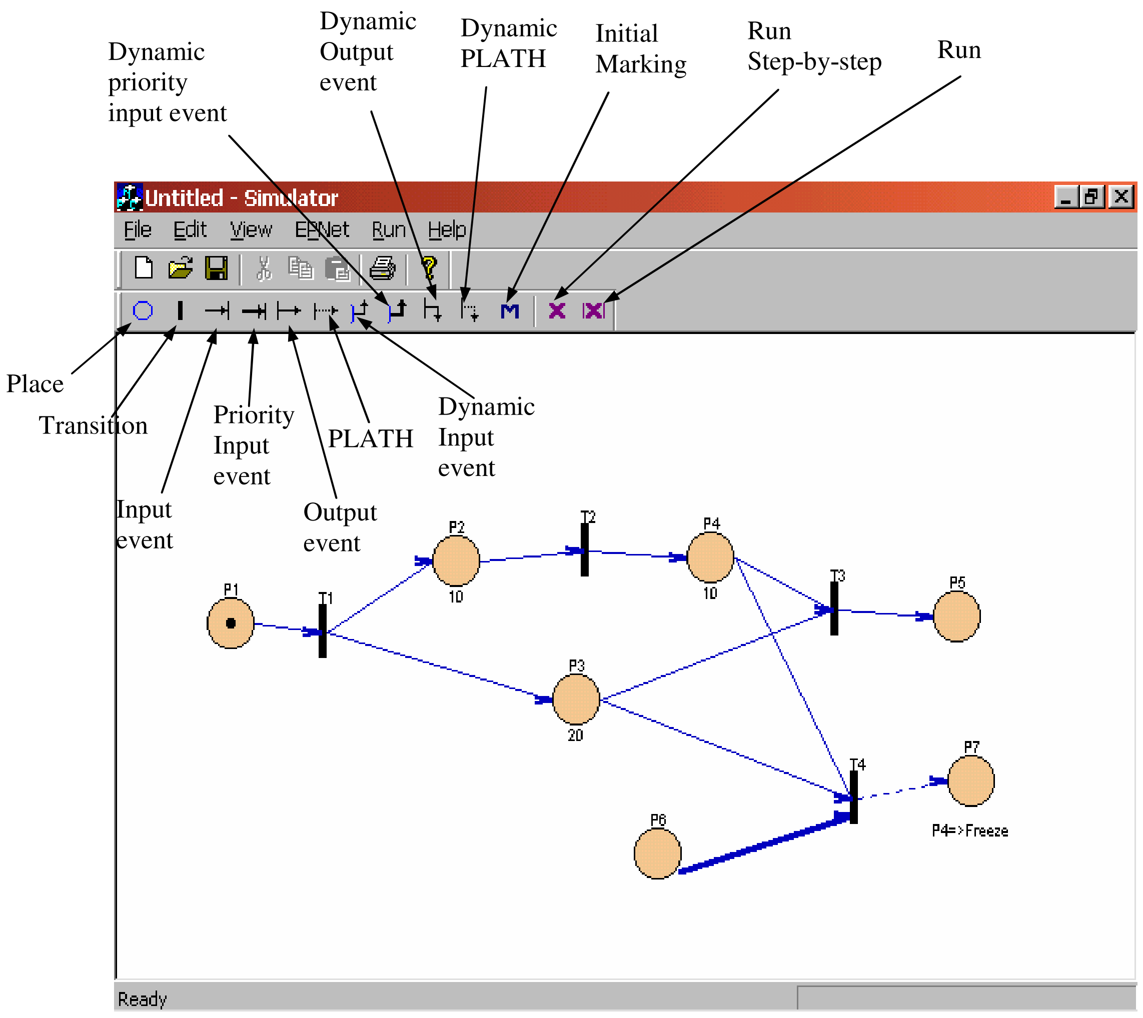
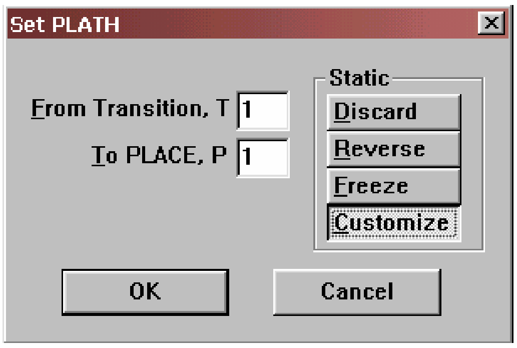


FIGURE 15. EP-net Simulator

5.2 HANDLING PLACES, PLATHS and DYNAMIC ARCS

Places in the net represent resources (e.g. audio playback, video playback etc). In this simulator, we are not restricting it to only those mentioned. Instead, we allow a place to be customized, i.e. it contains a customized resource specified by the user. Besides, the design of PLATH not only deals with the discard, reverse, freeze, restart and speed scaling operations, a customized operation can also be introduced. Figures 16 and 17 show the PLATH and dynamic output event menus respectively.

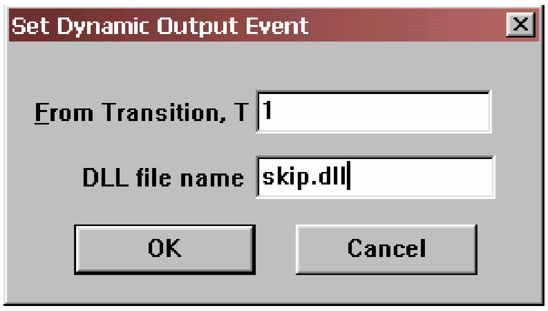
Dynamic link libraries (DLL) are used in the simulator. With DLL, the simulator links to the functionality in the library file when it runs. The library file remains a separate file that is referenced and called by the application. We can update and modify any functionality in the DLLs without having to recompile the application program, this is the reason why we have chosen DLL as our runtime library for the simulator. Therefore, to realize the places, PLATHs and dynamic arcs, we have employed DLLs to run their corresponding customized resources, customized operations and program statements.


Set PLATH

From Transition, T: 1

To PLACE, P: 1

FIGURE 16. PLATH menu


Set Dynamic Output Event

From Transition, T: 1

DLL file name: skip.dll

FIGURE 17. Dynamic output event menu

Visual C++ programming language was chosen because its C++ development environment and set of tools enable advanced applications for Windows and NT platforms to be created with ease.

5.3 Architecture of EP-Net Simulator

The simulator's architecture is demonstrated in figure 18. There are four main portions, namely: selection of medium and define their temporal relations, design initial marking, design the dynamic arcs and, compile and run the simulator.

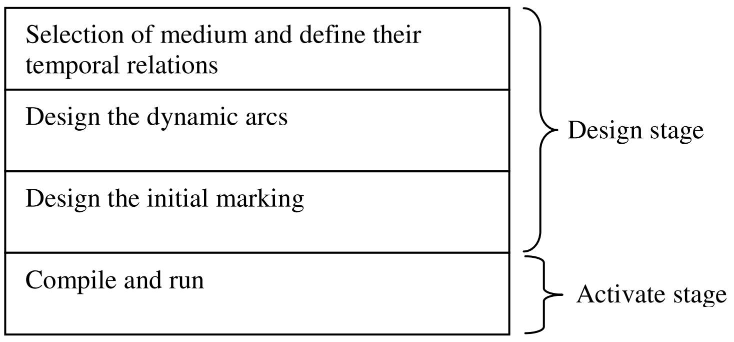

Selection of medium and define their temporal relations
Design the dynamic arcs
Design the initial marking
Compile and run
Design stage
Activate stage

FIGURE 18. State diagram of how the EP-Net Simulator operates

5.4 Implementation Issues

DLL Implementation for Dynamic Mechanisms

Initially, our implementation used different dynamic link library (DLL) for different dynamic mechanisms. For example, a dynamic output event carries out its skip task using the DLL file (skip.dll) as shown in figure 17. Hence, the DLL files grow with the number of dynamic mechanisms. In order to avoid such overhead, a DLL file is created which contains all program statements associated with its dynamic arcs used in the simulator. Each statement is given an ID called DLL Function ID as shown below.

```
switch(DLL Function ID)  {
        case 0: break;          //Not Used
        case 1: Program statements I;
                break;
        case 2: Program statements II;
                break;
        … …  }
```

PLATH Implementation Issues

One of the PLATH functions is to monitor and identify late arriving tokens, this is done by embedding a sequence number for each token in the system. A PLATH would record per incoming arc each arriving token's sequence number during execution. The stored sequence number(s) in a PLATH will be updated (i.e. incremented) whenever its transition is enabled and fires. With the PLATH sequence number(s) updated, the late-arriving token from each incoming arc which has been forced to fire previously can be captured.

6. CONCLUSIONS

In this paper we have presented an extended Prioritized Petri net, the enhanced Petri net (EP-net) to handle the side effect of the Prioritized Petri net (P-net). P-net holds a strong property, it will force firing events when the deadline is due regardless whether they are ready or not. This leads to a side effect known as premature/late arriving token problem. The paper has illustrated the problem and how EP-net can handle the problem by using the PLATH invoke pre-specified processing on premature/late arriving tokens.

In summary, we have studied the synchronization requirements for a multimedia presentation with user interactions. We use the concept of temporal relationship changes after interactions to understand the operations to be carried out, for various user inputs. We have proposed the EP-net model, which eliminated the shortcoming of P-net and increased the modeling power of OCPN in user interaction. A dynamic event mechanism has also been proposed for EP-net so that the target of an arc or PLATH can be decided at run-time. The *reverse* operation has been modeled using the EP-net features. Besides the *reverse* operation, other various user interactions e.g. *skip*, *freeze* and *restart*, and *speed scaling* operations can also be modeled by the EP-net model.

EP-net can also be applied when applications have tight and/or dynamic synchronization requirements. The illustrated multimedia UI handling is such an example. Memex-like applications can be another where users could frequently change trails when viewing through documents, whereby a change in the trail normally means a need to drop the viewing schedule of the current document and resynchronize to the new document viewing schedule. Dynamic arcs and PLATH will be useful to handle such condition changes. EP-net may also find applications in distributed control environments where run-time events frequently render changes to control sequences and synchronization conditions. PLATH would be useful here as it allows patching up the loopholes created from a forced priority firing, late arriving events can thus be handled. Dynamic arcs associated with program statements can be prescribed to anticipate such on-line control events and re-route control sequences accordingly.

In order to demonstrate the concepts are feasible, a prototype with runtime support has been developed (i.e. using visual C++ programming language running on Windows (NT) platform) and used to construct several interactive multimedia applications, including *skip*, *freeze* and *restart*, *reverse* and *speed scaling* operations.

REFERENCES

[1] Cosmos Nicolaou, “An Architecture for Real-Time Multimedia Communication Systems”, IEEE Journal on Selected Areas in Communications, Vol. 8, No. 3, pp.391-400, Apr. 1990.

[2] Fabio Bastian and Patrick Lenders, “Media Synchronization on Distributed Multimedia Systems”, International Conference on Multimedia Computing and System, pp. 526-531, 1994.

[3] Gerold Blakowski and Ralf Steinmetz, “A Media Synchronization Survey: Reference Model, Specification, and Case Studies”, IEEE Journal on Selected Areas in Communication, Vol. 14, No. 1, pp. 5-35, Jan. 1996.

[4] Chung-Ming Huang and Chung-Ming Lo, “An EFSM-Based Multimedia Synchronization Model and the Authoring System”, IEEE Journal on Selected Areas in Communication, Vol. 14, No. 1, pp. 138-152, Jan. 1996.

[5] Chung-Ming Huang and Chung-Ming Lo, “Synchronization for Interactive Multimedia Presentations”, IEEE Multimedia, Vol. 5, No. 4, pp. 44-62, 1998.

[6] Prabhat K., Andleigh and Kiran T., “Multimedia Systems Design”, Prentice Hall PTR, pp. 421-444, 1996.
[7] Guan Sheng-Uei, Yu Hsiao-Yeh, and Yang Jen-Shun, “A Prioritized Petri Net Model and Its Application in Distributed Multimedia Systems”, IEEE Transactions on Computers, Vol. 47, No. 4, pp. 477-481, Apr. 1998.
[8] Chung-Ming Huang, Chian Wang, and Cheng-Yi Kuo, “A Master-Medium-based Interactive Synchronization Control Scheme for Distributed Multimedia Systems, Euromicro Conference proceedings, Vol. 2, pp. 506-513, 1998.
[9] Jeffrey J. P. Tsai, Yaodong Bi, Steve J. H. Yang and Ross A. W. Smith, “Distributed Real-Time Systems”, A Wiley-Interscience Publication John Wiley & Sons Inc, Chapter 11, pp. 247-275, 1996.
[10] P. Lougher, “The Design of a Storage Server For Continuous Media”, IEEE Computer Journal, Vol. 36, 1993.
[11] Soon M Chung, “Multimedia Information Storage And Management”, Kluwer Academic Publishers, pp. 303-411, 1996.
[12] H. Thimm, W. Klas, “Managing Adaptive Presentation Executions in Distributed Multimedia Database System”, Proceedings 1996 International Workshop On Multimedia Database Management Systems, pp. 152-167, Aug 1996.
[13] G. Bruno, “Model-based Software Engineering”, Chapman and Hall, pp. 63-101, 1995.
[14] James L. Peterson, “Petri Net Theory and the Modeling of Systems”, Prentice-Hall, Inc, 1981.
[15] Thomas D. C. Little, “Synchronization and Storage Models for Multimedia Objects”, IEEE Journal on Selected Areas in Communication Vol. 8, No.3, pp. 413-427, April 1990.
[16] B. Prabhakaran and S. V. Raghavan, “Synchronization Models for Multimedia Presentation with User Participation”, ACM Multimedia proceedings, pp. 157-166, Aug. 1993.
[17] M. Woo, N.U. Qazi, and A. Ghafoor, “A Synchronization Framework for Communication of Pre-orchestrated Multimedia Information: IEEE Network, pp. 52-61, Feb. 1994.
[18] Kyoungro Yoon and P. Bruce Berra, “TOPCN: Interactive Temporal Model for Interactive Multimedia Documents”, International Workshop on Multimedia Database Management Systems, pp. 136-144, Aug. 1998.
[19] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franceschinis, “Modelling with Generalized Stochastic Petri Nets”, John Wiley & Sons, Chapter 3, pp. 49 – 68, 1996.
[20] Andrew F. Tanenbaum, “Computer Network”, Prentice Hall, pp. 219 - 239, 1996.
[21] Naveed U. Qazi, Miae Woo and Arif Ghafoor, “A Synchronization and Communication Model for Distributed Multimedia Objects”, Proceedings First ACM International Conference on Multimedia, pp. 147 - 155, Aug 1993.
[22] Michel Diaz and Patrick Senac, “Time Stream Petri Nets A Model for Multimedia Streams Synchronization”, Proceedings of the First International Conference on Multimedia Modelling, pp. 257-273, 1993.
[23] Yahya Y. Al-Salqan and Carl K. Chang, “Temporal Relations and Synchronization Agents”, IEEE Multimedia, Vol. 3., pp 30 - 39, 1996.
[24] Raghavan, S.V. Prabhakaran, B. Tripathi, S.K., "Synchronization representation and traffic source modeling in orchestrated presentation", IEEE Journal on Selected Areas in Communications, Vol. 14, No. 1, pp. 104 -113, Jan. 1996.