A Dynamic Petri Net Model for Iterative and Interactive Distributed Multimedia Presentation

Roy Tan and Sheng-Uei Guan

***Abstract*—Object Composition Petri Nets, Priority Petri Nets, Dynamic OCPN, and Enhanced P-Nets have extended the original Petri Net to achieve the modeling of media synchronization and asynchronous user interactions during multimedia playback. The dynamic Petri Net (DPN) has been conceptualized to tackle existing problems in these two areas of modeling distributed multimedia systems. DPN features dynamic modeling elements which allows iteration and hence is able to reduce graph sizes of synchronous playback models while allowing greater details to be shown. DPN also introduces asynchronous event handling techniques that are powerful and effective. DPN was used in the design and modeling of a multimedia orchestration tool which is a typical representation of an application that works in a distributed multimedia system.**

***Index Terms*—Dynamic events, interactive multimedia, iterative multimedia playback, multimedia synchronization, Petri net.**

I. Introduction

A. Current Trends

TODAY, development efforts in the computing world are geared toward providing large-scale distributed multimedia services through the Internet. These services include the distribution and transfer of digitized video and audio data. They will cater not only to the growing numbers of personal computers with Internet access, but also to the coming proliferation of Internet-enabled appliances such as video handphones and palmtops.

B. Multimedia Requirements

Many multimedia Internet services, such as video conferencing and video-on-demand, will require concurrent, real-time transfer of video and audio multimedia data. Since the Internet is a distributed environment, multimedia servers and clients will have to synchronize among themselves in order to perform effectively.

Many online multimedia systems rely on streaming as a means of distribution of playback media in real-time. In streaming, the media is usually segmented into smaller chunks and then sent to the client one by one. Thus, a model that can represent the playback of these segments effectively will be useful. Quite a few models [3]–[6], [8], [15], [16] have been developed in the past for this purpose; however, they lack the power of general programmability to model a multimedia presentation which is iterative in nature. The current status is that long segments of modeling elements need to be connected in long sequence with obvious similarity among them for this purpose.

Another feature of multimedia systems is allowing interruption of the flow of the playback at any time in the form of an asynchronous user input. This introduces asynchrony to the otherwise synchronous characteristic of multimedia playback. Therefore, models for these systems must also provide for asynchronous user inputs. Quite a few models [5], [6], [8], [15] have been developed with this feature in mind. The drawback as we can see is that most of them are ad hoc in nature. A model that is capable of modeling asynchronous user interrupts with general programmability mechanism in place will offer much richer modeling power for sophisticated multimedia applications.

C. Synchronization Models

Because multimedia distribution systems can be very complex, there is a need to model them for effective implementation. The Petri Net [1], [2] is commonly used to model concurrent systems. Extensions have been made to the Petri Net to improve its functionality in representing multimedia synchronization. These include Object Composition Petri Net (OCPN) [3], Extended OCPN (XOCPN) [4], Prioritized Petri Net (P-Net), Distributed OCPN (DOCPN) [5], and Enhanced Prioritized Petri Net (EP-Net) [6].

This paper introduces a new extension to the Petri Net model. The Dynamic Petri Net (DPN) is a powerful extension to the Petri Net family. DPN adds several new elements such as dynamic places, control places, control functions, control output arcs, and control variables. Control variables may be modified dynamically by control functions which are represented in pseudocode. This allows a system with an indeterminate number of repetitive states to be modeled.

Control output arcs may be enabled and disabled according to the values of control variables. This endows DPN with the ability of modeling asynchrony. DPN also introduces iteration into the Petri Net model, making it possible to reduce graph size while retaining detail.

DPN is well suited to the modeling of distributed multimedia systems as such systems require both the representation of many repetitive states (playback of video and audio segments) and also the representation of asynchronous user interrupts such as skip, freeze, restart, forward, and reverse.

Manuscript received June 6, 2000; revised April 6, 2004. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jeff A. Bilmes.

R. Tan is at 1 Yishun Ave 7, Singapore 768923 (e-mail: roy-wk_tan@agilent.com).

S.-U. Guan is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119260 (e-mail: eleguans@nus.edu.sg).

Digital Object Identifier 10.1109/TMM.2005.854377

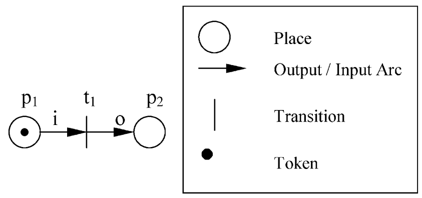


Fig. 1. Petri Net segment.

D. Orchestration Tool Implementation

A multimedia distribution system has been modeled and implemented to showcase the modeling functionality and power of the DPN. A multimedia orchestration tool was chosen as a typical representation of a application that works in a distributed multimedia system environment which synchronizes playback and processes asynchronous user inputs.

II. Related Work

The DPN is based on previous work done on the original Petri Net [1], [2], [14], OCPN [3], XOCPN [4], P-Net [5], DOCPN [5], and EP-Net [6].

A. Petri Net

Petri Nets were first conceptualized by Carl Adam Petri in a seminal paper, *Kommunikation mit Automaten* in 1962. The model was later refined by Anatol Holt and given its present name. There has been a steadily increasing interest in Petri Nets after this because of the Petri Net's ability to represent both concurrency and nondeterminacy [14]. Twenty-five years of theoretical and practical work and several thousand research papers have proven that Petri Nets are one of the most useful languages available for modeling concurrent processes.

A Petri Net structure, P, is a quadruple.

1) $\mathrm{P} = \{\mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_\mathrm{x}\}$, where $\mathrm{x} \geq 0$, is a finite set of *Places*.
2) $\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots \mathrm{t}_\mathrm{y}\}$, where $\mathrm{y} \geq 0$, is a finite set of *Transitions*. where $\mathrm{P} \cap \mathrm{T} = \emptyset$ i.e., the set of the places and transitions are disjoint.
3) I: $\mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *Input Arc*, a mapping from places to bags of transitions.
4) O: $\mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *Output Arc*, a mapping from transitions to bags of places.

$\mathrm{Token} = \{\mathrm{token}_1, \mathrm{token}_2, \ldots \mathrm{token}_\mathrm{x}\}, \mathrm{x} \geq 0, \mathrm{x} \in \Im$, is a finite set of dynamic markings on places.

The Petri Net model consists of places, transitions, arcs, and tokens.

1) A *place*, denoted by a circle, represents the state of the system. p_1 and p_2 in Fig. 1 are places.
2) A *transition*, denoted by a vertical line, represents the action of the system and is led by an output arc and trailed by an input arc. t_1 in Fig. 1, led by o and trailed by i, is a transition.
3) An *input arc*, denoted by an arc terminated by an arrowhead leading from a place to a transition, maps a place to a transition. i in Fig. 1 is an input arc.

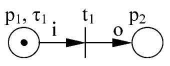


Fig. 2. OCPN segment.

4) An *output arc*, denoted by an arc terminated by an arrowhead leading from a transition to a place, maps a transition to a place. o in Fig. 1 is an output arc.
5) A *token* is a marking that denote the current state of the system. A firing of a transition removes a token from its input place and places a token in its output place. In Fig. 1, a token is marked in place, p_1.
6) The *input place of a transition* is the place that is connected to the transition via an input arc.
7) The *output place of a transition* is the place that is connected to the transition via an output arc.

The Petri Net is governed by a set of Firing Rules that allows movement from one state to another.

1) A transition is *enabled* when all input places that are connected to it via an input arc have at least one token.
2) A firing of a transition removes a token from its input place and places a token in its output place.

B. OCPN and XOCPN

An OCPN structure, OCPN, is a 5-tuple.

$\mathrm{OCPN} = (\mathrm{P}, \mathrm{T}, \mathrm{I}, \mathrm{O}, \tau)$.

1) $\mathrm{P} = \{\mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_\mathrm{x}\}$, where $\mathrm{x} \geq 0$ is a finite set of *Places*.
2) $\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots \mathrm{t}_\mathrm{y}\}$, where $\mathrm{y} \geq 0$ is a finite set of *Transitions*, where $\mathrm{P} \cap \mathrm{T} = \emptyset$, i.e., the set of the places and transitions are disjoint.
3) I: $\mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *Input Arc*, a mapping from transitions to bags of places.
4) O: $\mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *Output Arc*, a mapping from transitions to bags of places.
5) $\tau = \{\tau_1, \tau_2 \ldots \tau_\alpha\}$, where $\alpha \geq 0$ is a finite set of *Time Intervals* representing playback time intervals. This is derived from OCPN.

The OCPN model adds the element of time intervals to the original Petri Net. It considers places as objects and attaches time intervals to them. These time intervals are typically represented beside place identifiers as in Fig. 2 for p_1. A place which is associated with a time interval is called a *timed place*.

The original Petri Net firing rules are modified for all transitions that have an input timed place.

1) When a token is created in a timed place, the token is *locked* for the duration of the time interval specified. The token becomes *unlocked* when the duration specified is over.
2) A token created in a nontimed place, or a place not associated to a timed interval, is deemed to be *unlocked* at all times.
3) A transition is *enabled* if the all of its input places contain at least one unlocked token.

XOCPN applies granulation of the objects or places and essentially allows finer time-intervals or Synchronization Interval Units, for synchronization.

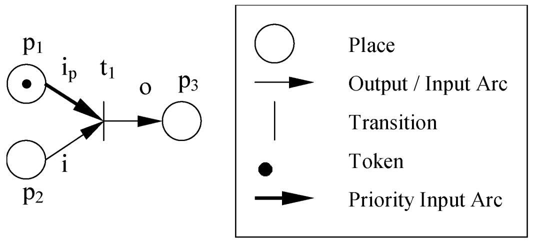


Fig. 3. P-Net segment.

C. P-Net, DOCPN, and EP-Net

The P-Net family of modeling structures is a family of Petri Net modeling structures based on the addition of the Priority Arc. It includes P-Net, Distributed OCPN, and Enhanced P-Net.

P-Net was introduced by Guan *et al.* [5] to allow Petri Nets to model asynchronous systems with user interrupt events.

DOCPN is an addition to the P-Net to increase the functionality of the P-Net by merging it with the OCPN and XOCPN models mentioned in II.B.

EP-Net was proposed later to handle late and/or premature tokens due to the limitations of P-Net [6].

A P-Net structure, P-Net, is a 5-tuple.

$\mathrm{P{-}NET} = (\mathrm{P}, \mathrm{T}, \mathrm{I}, \mathrm{O}, \mathrm{Ip})$.

1) $\mathrm{P} = \{\mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_\mathrm{x}\}$, where $\mathrm{x} \geq 0$ is a finite set of *Places*.
2) $\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots \mathrm{t}_\mathrm{y}\}$, where $\mathrm{y} \geq 0$ is a finite set of *Transitions*, where $\mathrm{P} \cap \mathrm{T} = \emptyset$, i.e., the set of the places and transitions are disjoint.
3) $\mathrm{I}\colon \mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *Input Arc*, a mapping from transitions to bags of places.
4) $\mathrm{O}\colon \mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *Output Arc*, a mapping from transitions to bags of places.
5) $\mathrm{I}_\mathrm{p}\colon \mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *Priority Input Function*, a mapping from transitions to bags of places.

P-Net adds an additional element to the original Petri Net. This is the priority input arc. The priority input arc shown in Fig. 3 is i_p, connecting p_1 to t_1.

The firing rules are modified for a P-Net.

1) A transition is enabled if there is at least one token in an input place that is connected to the transition via a priority input arc regardless of whether there are tokens in any input place that are connected to the transition via a nonpriority input arc.
2) However, if there are two or more input places that are connected to the transition via priority input arcs, the transition is only enabled when all such input places have at least one token each. This is also known as the "AND" rule.

An example of the P-Net is in the segment shown in Fig. 3. A token has been created in place p_1. As the input arc i_p is a priority arc (bold), transition t_1 is enabled even though p_2 has no token. Hence, the output arc o will fire and a token will be created in p_3.

The Dynamic OCPN model is the P-Net model that utilizes the characteristics of OCPN and XOCPN. It is essentially a P-Net that allows temporal information to be associated to its places.

EP-Net added various new elements such as dynamic input and output arcs in a bid to model asynchrony. These use programming statements to dynamically redirect arcs to places and transition.

III. Dynamic Petri Net

A. Definitions

A Dynamic Petri Net structure, S, is a 10-tuple.

$\mathrm{S} = (\mathrm{P}, \mathrm{T}, \mathrm{I}, \mathrm{O}, \tau, \mathrm{P_d}\{\mathrm{F}\}, \mathrm{N}, \mathrm{F}, \mathrm{P}\{\mathrm{F}\}, \mathrm{O_c}\{\mathrm{F}\})$

1) $\mathrm{P} = \{\mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_\mathrm{x}\}$, where $\mathrm{x} \geq 0$ is a finite set of *places*.
2) $\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots \mathrm{t}_\mathrm{y}\}$, where $\mathrm{y} \geq 0$ is a finite set of *transitions*, where $\mathrm{P} \cap \mathrm{T} = \emptyset$, i.e., the set of the places and transitions are disjoint.
3) $\mathrm{I}\colon \mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *input arc*, a mapping from transitions to bags of places.
4) $\mathrm{O}\colon \mathrm{T} \rightarrow \mathrm{P}^\infty$ is the *output arc*, a mapping from transitions to bags of places.
5) $\tau = \{\tau_1, \tau_2, \ldots \tau_\alpha\}$, where $\alpha \geq 0$ is a finite set of *time intervals* representing playback time intervals. This is derived from OCPN.
6) $\mathrm{N} = \{\mathrm{n}_1, \mathrm{n}_2, \ldots \mathrm{n}_\beta\}$, where $\beta \geq 0$ is a finite set of persistent *control variables*. These variables are *persistent* through *every marking* of the net.
7) $\mathrm{F} = \{\mathrm{f}_0, \mathrm{f}_1, \ldots, \mathrm{f}_\gamma\}$, where $\gamma \geq 0$ is a finite set of *control functions* that perform functions based on any control variable N.
8) $\mathrm{P}\{\mathrm{F}\}\colon \mathrm{P}\{\mathrm{F}\} \subseteq \mathrm{P}$ is a finite set of (static) *control places* (a subset of P) that executes any control function F.
9) $\mathrm{O^c}\{\mathrm{F}\}\colon \mathrm{O^c} \subseteq \mathrm{O}$ is a finite set of (static) *control output arcs* that may be disabled or enabled according to any control function F.
10) $\mathrm{P^d}\{\mathrm{F}\}\colon \mathrm{P^d} \subseteq \mathrm{P}$ is a finite set of *dynamic places* (a subset of P) that takes their value from some control function F.

$\mathrm{Token} = \{\mathrm{token}_1, \mathrm{token}_2, \ldots \mathrm{token}_z\}, \mathrm{z} \geq 0$, is a finite set of *dynamic markings* on places.

The DPN model adds a number of new elements to the original Petri Net, as well as inheriting some from its extensions.

1) *Time Intervals* were inherited from the DOCPN model. They are associated with a place and are written beside the place identifier (e.g., $p_a\tau_a$ in Fig. 4).
2) *Dynamic places* are places that are variable and change according to the control function. They are identified by p^d. The control function is stated within the curly parentheses { } beside the place identifier (e.g., $p_z^d\{\text{is\_zero}\}$ in Fig. 4). To be more precise, a dynamic place is a place variable that actually becomes the place that it is assigned to. A dynamic place changes according to its control function only when a token is created in that place.
3) *Control variables* are a set of pre-defined integer variables that have certain relationship to the system being modeled. They hold their values throughout every marking of the net. They may only be modified by control functions. They are typically initialized by an init function at the start place.
4) *Control functions* are a set of pre-defined functions which act on control variables in a way determined by some pseudo-code statements. C-style pseudo-code is used for defining these statements. Note that it is crucial that the pseudo-code for a control function is implemented

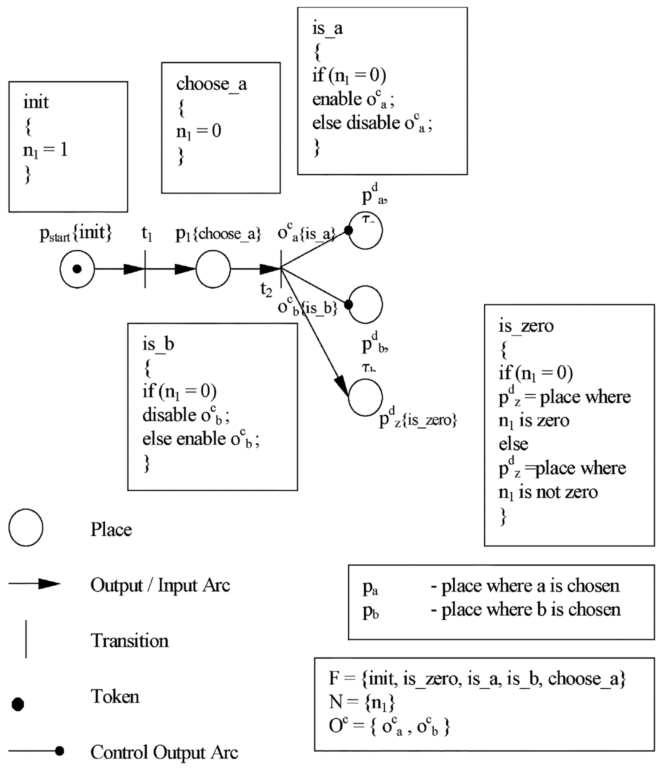

init { n1 = 1 }
choose_a { n1 = 0 }
is_a { if (n1 = 0) enable o^c_a ; else disable o^c_a ; }
p_start{init}
t1
p1{choose_a}
o^c_a{is_a}
p^d_a
t2
o^c_b{is_b}
p^d_b
p^d_z{is_zero}
is_b { if (n1 = 0) disable o^c_b ; else enable o^c_b ; }
is_zero { if (n1 = 0) p^d_z = place where n1 is zero else p^d_z =place where n1 is not zero }
Place
Output / Input Arc
Transition
Token
Control Output Arc
pa - place where a is chosen
pb - place where b is chosen
F = {init, is_zero, is_a, is_b, choose_a}
N = {n1}
O^c = { o^c_a , o^c_b }

Fig. 4. Typical DPN.

correctly, otherwise, it may cause a DPN to perform erratically due to a function bug or defect. In Fig. 4, the functions are defined in text boxes. Control functions are used by control places, dynamic places, and control output arcs. To avoid ambiguity, control functions follow this rule. For any marking, any control function with conditional statements will be executed after control functions without conditional statements are executed. Furthermore, the control function of a control output arc is executed after all other control functions have been executed.

5) *Control places* are a subset of Petri Net places. They are associated with a control function by which they control the value of the control variable. Control places are identified by a place identifier followed by the control function identifier in curly parentheses (e.g., $p_1\{\text{choose\_a}\}$ in Fig. 4). The control functions will be executed only when a token has been created in the place.
6) *Control output arcs* are a subset of Petri Net Output Arcs. They are associated with a control function which disables or enables the output arc. It is represented by an arc terminated with a circle. Control output arcs are identified by o^c followed by the control function identifier in curly parentheses (e.g., $o^c_a\{\text{is\_}a\}$ and $o^c_b\{\text{is\_}b\}$ in Fig. 4). The conditions for enabling and disabling are checked at each marking. These control functions are executed after all other control functions have been executed.

The firing rules of the DPN are modified only for the control output arcs. An enabled transition will only fire when it is enabled (i.e., when it is *not* disabled).

The priority rules for execution of control functions for each marking are as follows.

1) Control functions without conditional statements have the highest priority and are executed first.
2) Control functions with conditional statements are executed next.
3) Control functions of control output arcs are of lowest priority and are executed last.

Fig. 4 showcases the abilities and features of DPN. Upon starting, the control function init initializes n_1 to 1. Transition t_1 is subsequently enabled. A token is subsequently created in p_1. In p_1, the control function $\text{choose\_}a$ changes the value of the control variable n_1 to 0. This causes o^c_a to be enabled and and o^c_b to be disabled. Transition t_2 is enabled. Since o^c_a is enabled and o^c_b is disabled, a token is only created in p_a and p^d_z. Dynamic place p^d_z translates to places where n_1 is 0 according to control function is_zero.

B. DPN Design Requirements and Discussions

1) Streaming Multimedia Playback Requirements: Streaming multimedia playback is sometimes tricky to model as media segmentation may differ in different systems. Moreover, there will be many media segments to model. A traditional method of modeling multimedia segments is to model each segment playback individually as a place. If there were, for example, 100 segments of media, the model would have been too large as the size of the Petri Net increases linearly with the number of media.

Another way would be to represent all the media as a monolithic block of media and doing away with segments altogether. This would reduce the graph size but significantly lower the level of detail.

Furthermore, for both the above methods, representing a jump or change in media would not be feasible. Therefore, a method is needed to allow both a reduction in graph size and an increase in the level of detail, representation and modeling.

2) Priority Arcs: The Priority Arc of P-Net and EP-Net were initially considered for use in representing asynchrony in the Dynamic Petri Net model. However, control places and control output arcs were used instead as they were able to represent asynchrony as well as provide iterative power.

3) Iterative Ability of DPN: It was found that to model segmented media effectively, there is a need to provide an iterative ability. DPN was thus designed with control variables that may be used for iteration. Control functions provide the ability to modify these variables and determine dynamic places that depend on these iterated variables. It was found that DPN was able to model the playback of segmented media effectively so that all segments of the media need not be represented individually but may be represented iteratively, allowing a greater level of detail without the cost of an explosion in graph size. Dynamic arcs as used in EP-Net are not enough to provide an iterative ability. An iterative ability does not only require looping but also requires an exit condition for loop control.

4) Asynchronous Ability of DPN: Control output arcs were introduced to DPN to provide the level of asynchronous mod-

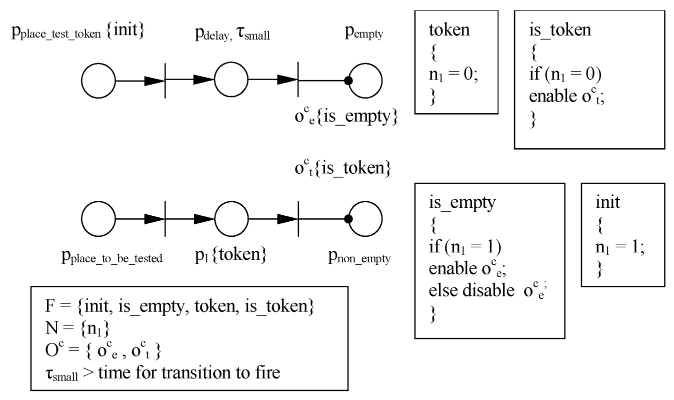

p_place_test_token {init}
p_delay, τ_small
p_empty
o^c_e{is_empty}
o^c_t{is_token}
p_place_to_be_tested
p_1{token}
p_non_empty
token { n_1 = 0; }
is_token { if (n_1 = 0) enable o^c_t; }
is_empty { if (n_1 = 1) enable o^c_e; else disable o^c_e; }
init { n_1 = 1; }
F = {init, is_empty, token, is_token}
N = {n_1}
O^c = { o^c_e , o^c_t }
τ_small > time for transition to fire

Fig. 5. Turing machine emulation.

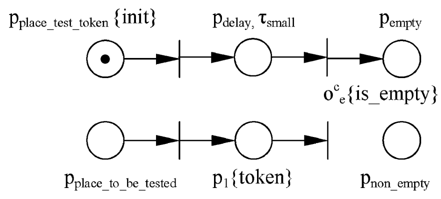


Fig. 6. Case 1: 0th marking.

eling that P-Net and EP-Net enjoyed through the use of the priority arc.

C. Turing Machine Modeling

The control output arcs were able to avoid the Late Arriving Token problem encountered using priority arcs. DPN was also found able to emulate a Turing Machine, proving that it has at least the same mechanism of asynchronous ability as P-Nets and EP-Nets.

Theorem: DPN can emulate a Turing Machine. It has been shown that an extended Petri Net model with the ability to test a place for zero token can emulate a Turing Machine [5], [7]. The ability to test a place for zero token is an indication that the model has the ability to perform a conditional statement like "if ... then ... else", which is missing in the original Petri Net model. In the following we show DPN has the ability to test a place for zero token.

Proof: To prove the theorem, we consider the DPN in Fig. 5. There may be two possible cases.

Case 1— The system starts with a token in $p_{\text{place\_test\_token}}$.

Case 2— The system starts with a token in $p_{\text{place\_test\_token}}$ and a token in $p_{\text{place\_to\_be\_tested}}$. In case 1, it is expected that a token will be created in p_{empty} and no token will be created in $p_{\text{non\_empty}}$.

In case 2, it is expected that a token will be created in $p_{\text{non\_empty}}$ and no token will be created in p_{empty}.

τ_{small} is assumed to be finite and nonzero.

Fig. 6 shows the initial marking of case 1. There is no token in $p_{\text{place\_test\_token}}$, the control function *init* sets n_1 to 1, hence the control output arc o_t^c is disabled.

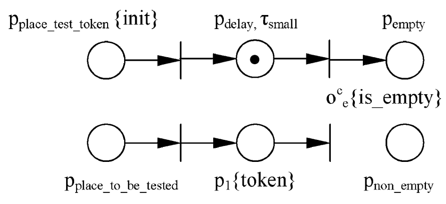


Fig. 7. Case 1: 1st marking.

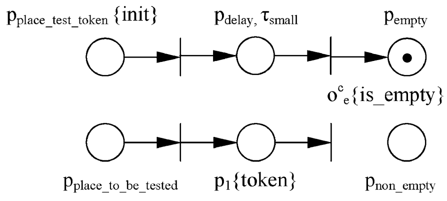


Fig. 8. Case 1: 2nd marking.

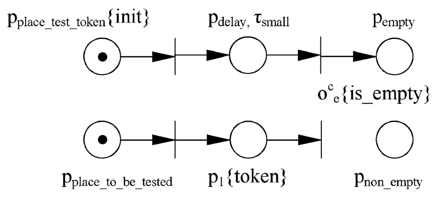


Fig. 9. Case 2: 0th marking.

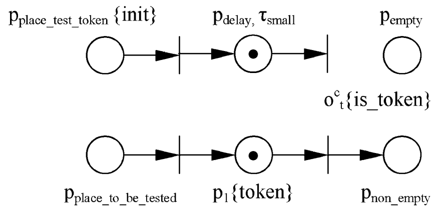


Fig. 10. Case 2: 1st marking.

The next marking shown in Fig. 7 shows a token created in p_{delay}. The time interval, τ_{small}, plays no significant part in case 1. There are no changes to the control variable n_1, hence there are no changes to either of the output arcs.

The final marking shown in Fig. 8 shows a token created in p_{empty} as a consequence of the positioning of a token in p_{delay} in previous marking. Hence, the theorem is partially proven.

Fig. 9 shows the initial marking of case 2. There is a token in $p_{\text{place\_test\_token}}$ and in $p_{\text{place\_to\_be\_tested}}$. The control function *init* sets n_1 to 1 and causes the control output arc o_t^c to be disabled as in Fig. 7 as $p_{\text{place\_to\_be\_tested}}$ does not invoke any control function.

In the next marking shown in Fig. 10, a token is created in p_{delay} and p_1. The control function *token* is invoked to set n_1 to 0. The control functions of is_token and is_empty are in invoked to disable o_e^c and enable o_t^c. The delay afforded by τ_{small} allows the function *token* to modify the control variable before a token is created in p_{empty}. It removes any concurrency issues that may arise whether is_empty or *token* is invoked first which may lead to ambiguity.

The final marking shown in Fig. 11 shows a token created in $p_{\text{non\_empty}}$ as a consequence of the positioning of a token in p_1

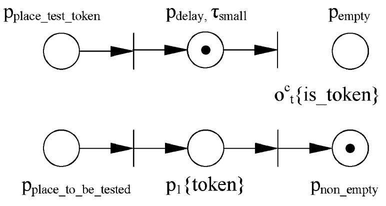
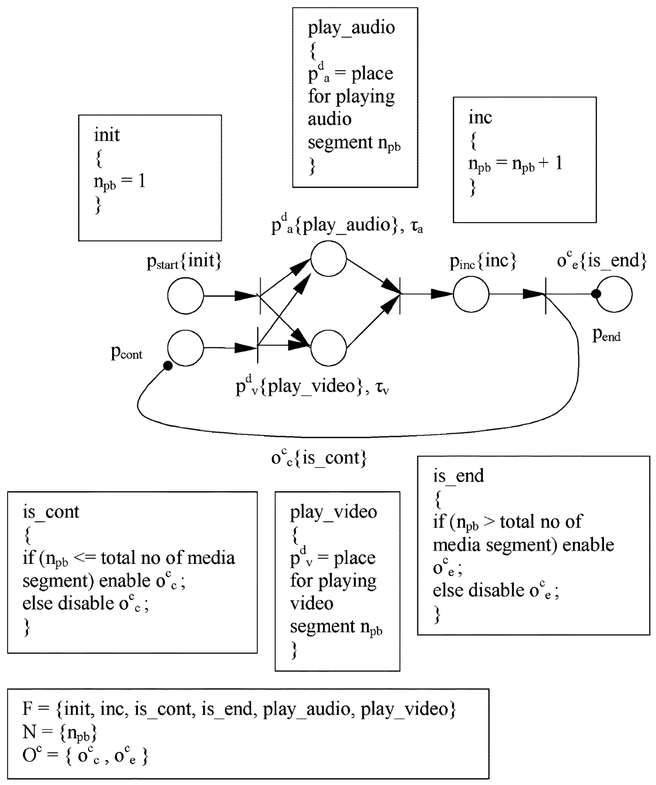


Fig. 11. Case 2—2nd marking.



```
init
{
n_pb = 1
}

play_audio
{
p^d_a = place
for playing
audio
segment n_pb
}

inc
{
n_pb = n_pb + 1
}

is_cont
{
if (n_pb <= total no of media
segment) enable o^c_c ;
else disable o^c_c ;
}

play_video
{
p^d_v = place
for playing
video
segment n_pb
}

is_end
{
if (n_pb > total no of
media segment) enable
o^c_e ;
else disable o^c_e ;
}

F = {init, inc, is_cont, is_end, play_audio, play_video}
N = {n_pb}
O^c = { o^c_c , o^c_e }
```

Fig. 12. DPN model of streaming media playback.

in previous marking. Hence, the theorem is proved. Therefore, DPN can emulate Turing machines. □

Discussions: Now that the DPN model has the power of Turing machines, the issues of reachability and liveness [14] are undecidable. However, for applications that have DPNs free from cycles (which could be formed from the use of control functions and dynamic places), it will be live, i.e., free from deadlocks. A Petri net is conservative if the number of tokens in the net is conserved. For DPN, the number of tokens in a net will depend on the concurrency of the modeled system and, therefore is not conservative.

In modular or hierarchical systems, DPNs may need to be composed or embedded while doing so is not quite clear due to the use of control variables/functions and dynamic places. To ensure a straightforward composition or embedding, the target for a dynamic place may need to be constrained such that it will not fall outside of the current layer or scope.

Finally, it should be mentioned there exists some relation between DPN and the Colored Petri-net (CPN) [2] due to the use of control functions in DPN is similar to the use of labeled arcs and transition guards in CPN.

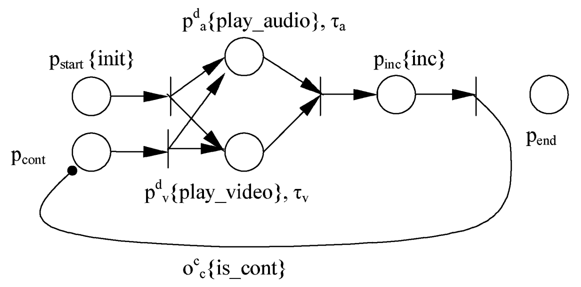


Fig. 13. The 0th marking: streaming media playback.

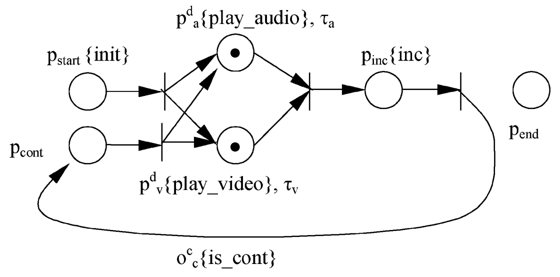


Fig. 14. The 1st marking: streaming media playback.

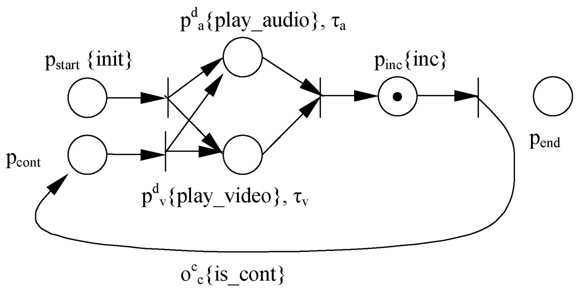


Fig. 15. The 2nd marking: streaming media playback.

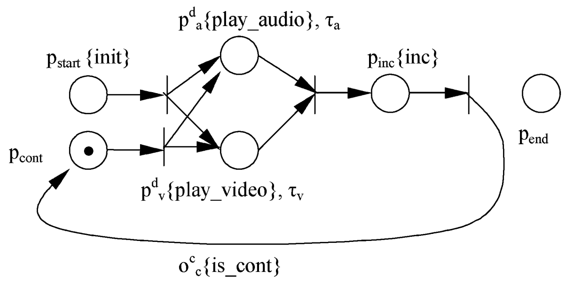


Fig. 16. The 3nd marking: streaming media playback.

D. DPN Applications

1) Modeling Streaming Multimedia Arcs: Consider a media playback system of an indeterminate number, n, of media resources. This system is modeled as shown in Fig. 12. Upon starting shown in Fig. 13, n_{pb} is set to 1 via function *init*. Arc o_e^c is disabled and o_c^c is enabled. On the next transition shown in Fig. 14, the dynamic places for playing audio and video plays segment 1 of each resource. In Fig. 15, a token is now in p_{inc}. The *inc* control function increments the value of n_{pb} by 1. Assuming that number of media segments is greater than 1, o_c^c remains enabled and o_e^c remains disabled. Thus, a token will be created in p_{cont} in the next marking. In Fig. 16, it is easy to see

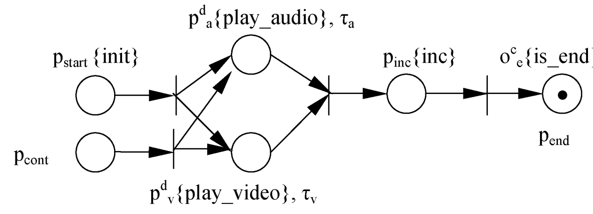
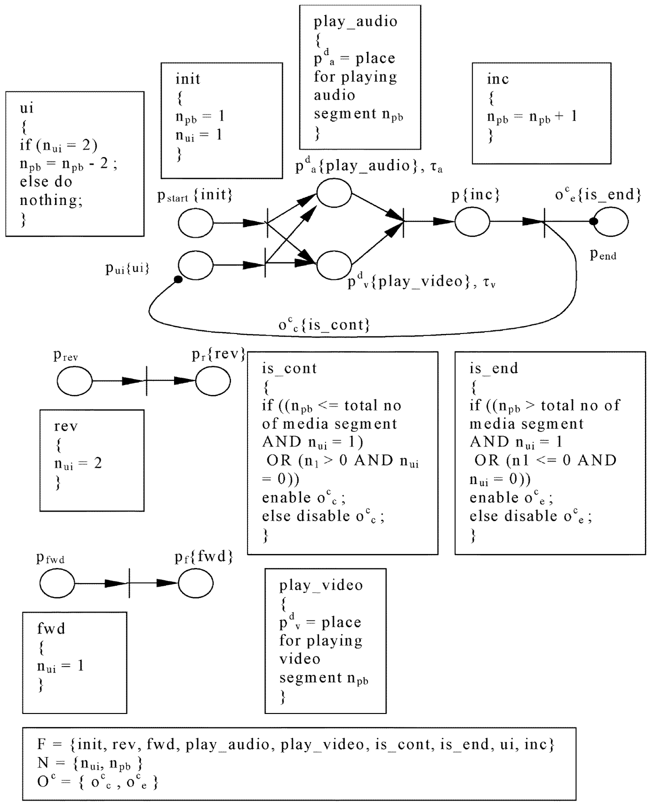


Fig. 17. Final marking: streaming media playback.



```
ui
{
if (n_ui = 2)
n_pb = n_pb - 2 ;
else do
nothing;
}
```

```
init
{
n_pb = 1
n_ui = 1
}
```

```
play_audio
{
p^d_a = place
for playing
audio
segment n_pb
}
```

```
inc
{
n_pb = n_pb + 1
}
```

```
rev
{
n_ui = 2
}
```

```
is_cont
{
if ((n_pb <= total no
of media segment
AND n_ui = 1)
 OR (n_l > 0 AND n_ui
= 0))
enable o^c_c ;
else disable o^c_c ;
}
```

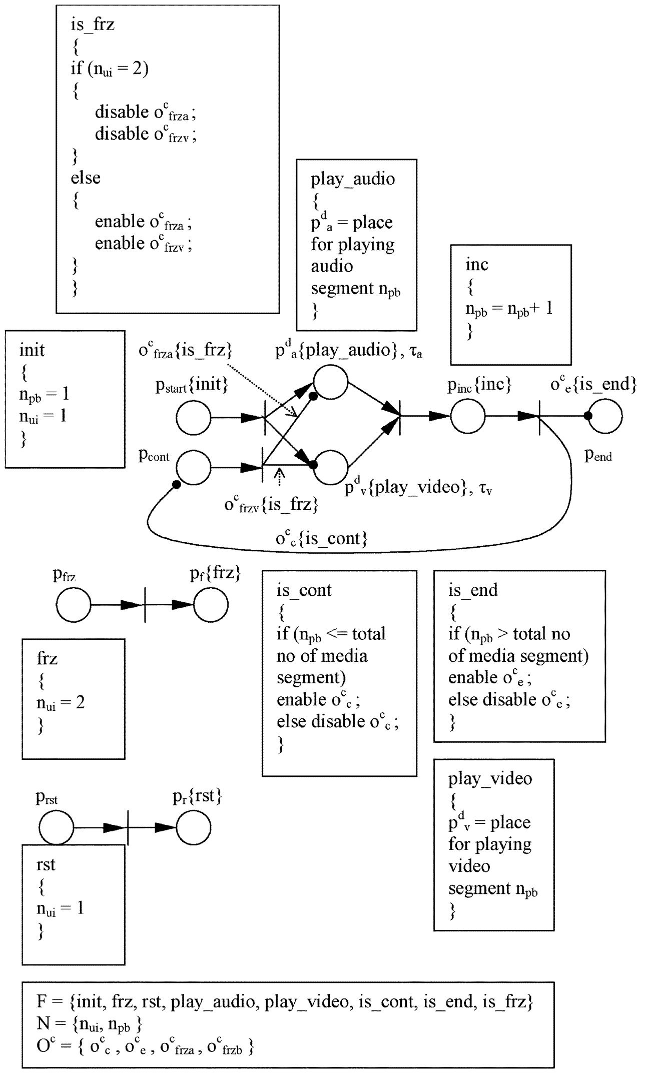
```
is_end
{
if ((n_pb > total no of
media segment
AND n_ui = 1
 OR (nl <= 0 AND
n_ui = 0))
enable o^c_e ;
else disable o^c_e ;
}
```

```
fwd
{
n_ui = 1
}
```

```
play_video
{
p^d_v = place
for playing
video
segment n_pb
}
```

```
F = {init, rev, fwd, play_audio, play_video, is_cont, is_end, ui, inc}
N = {n_ui, n_pb }
O^c = { o^c_c , o^c_e }
```

Fig. 18. Multimedia playback with reverse and forward user interrupts.



```
is_frz
{
if (n_ui = 2)
{
   disable o^c_frza ;
   disable o^c_frzv ;
}
else
{
   enable o^c_frza ;
   enable o^c_frzv ;
}
}
```

```
play_audio
{
p^d_a = place
for playing
audio
segment n_pb
}
```

```
inc
{
n_pb = n_pb+ 1
}
```

```
init
{
n_pb = 1
n_ui = 1
}
```

```
frz
{
n_ui = 2
}
```

```
is_cont
{
if (n_pb <= total
no of media
segment)
enable o^c_c ;
else disable o^c_c ;
}
```

```
is_end
{
if (n_pb > total no
of media segment)
enable o^c_e ;
else disable o^c_e ;
}
```

```
rst
{
n_ui = 1
}
```

```
play_video
{
p^d_v = place
for playing
video
segment n_pb
}
```

```
F = {init, frz, rst, play_audio, play_video, is_cont, is_end, is_frz}
N = {n_ui, n_pb }
O^c = { o^c_c , o^c_e , o^c_frza , o^c_frzb }
```

Fig. 19. Multimedia playback with freeze and restart user interrupts.

that $p_{\rm cont}$ allows the playback to continue. Now that $n_{\rm pb}$ is incremented to 2, the places p_v^d and p_a^d will play media segments 2 next. The playback goes on until $n_{\rm pb}$ is incremented beyond the number of the total media segments by the function *inc*. In this case, function is_end enables o_e^c and disables o_c^c and thus halting and ending playback shown in Fig. 17.

2) Modeling Reverse and Forward: In Fig. 18, assume that $p_{\rm rev}$ gets a token when the "reverse" user interaction is invoked. $p_{\rm fwd}$ gets a token when the "forward" user interaction is invoked. $p_{\rm cont}$ in the model in the previous section is replaced by the control function $p_{\rm ui}$. Playback initially is normal as all control variables are initialized to 1. The function *ui* does nothing when $n_{\rm ui}$ is 1 so that the playback proceeds.

Consider the case when "reverse" is invoked. The control place p_r calls the control function rev which changes $n_{\rm ui}$ to 2. As the playback continues, the system will reach $p_{\rm ui}$. This time, the control function will cause $n_{\rm ui}$ to be decremented by 2. This is to offset the previous increment by the control function inc. The combined result is that $n_{\rm ui}$ will be decremented by 1 each time. This will effect a reverse playback. Reverse playback ends when the number of media segments reaches zero.

Now, consider the case when "forward" is invoked. The control place p_f calls the control function *forward* which changes $n_{\rm ui}$ to 1. The next time the system is at $p_{\rm ui}$, no operation will be performed, thus effecting normal, forward playback.

3) Modeling Freeze and Restart: Once again in Fig. 19, the initial playback is normal as $n_{\rm ui}$ is initialized to 1. This enables the control output arcs $o_{\rm frza}^c$ and $o_{\rm frzv}^c$, allowing normal playback. When a "freeze" is invoked, $p_{\rm frz}$ will receive a token, in turn creating a token in the control place pf. The control function frz sets $n_{\rm ui}$ to 2 and disables $o_{\rm frz}^c$ and $o_{\rm frz2}^c$. Hence, when a token next arrives at $p_{\rm cont}$, it will be held there, waiting as there are no output arcs to fire. This effects a freeze in media playback. When a "restart" is invoked, control function rst sets $n_{\rm ui}$ back to 1, enabling the control output arcs $o_{\rm frza}^c$ and $o_{\rm frzv}^c$, causing playback to resume as normal.

4) Modeling Skip: In Fig. 20, the initial playback is again as normal. The control function *ui* acts on $n_{\rm ui}$, which has been initialized to 1, and does nothing, allowing normal playback.

The invocation of the "skip" user interaction causes the control function *skip* to be invoked via control place $p_{\rm skip}$. The function skip does two things. It first changes the value of $n_{\rm ui}$ from 1 to 2, then it sets the value of $n_{\rm skip}$ to the media segment number

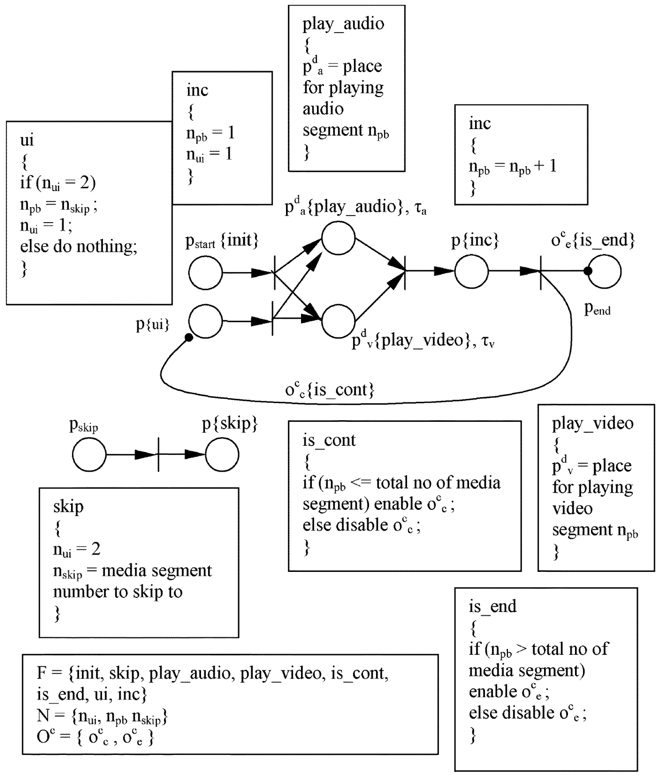

ui { if (n_ui = 2) n_pb = n_skip ; n_ui = 1; else do nothing; }
inc { n_pb = 1 n_ui = 1 }
play_audio { p^d_a = place for playing audio segment n_pb }
inc { n_pb = n_pb + 1 }
p^d_a{play_audio}, τ_a
p_start{init}
p{inc}
o^c_e{is_end}
p{ui}
p^d_v{play_video}, τ_v
p_end
o^c_c{is_cont}
p_skip
p{skip}
is_cont { if (n_pb <= total no of media segment) enable o^c_c ; else disable o^c_c ; }
play_video { p^d_v = place for playing video segment n_pb }
skip { n_ui = 2 n_skip = media segment number to skip to }
is_end { if (n_pb > total no of media segment) enable o^c_e ; else disable o^c_e ; }
F = {init, skip, play_audio, play_video, is_cont, is_end, ui, inc}
N = {n_ui, n_pb n_skip}
O^c = { o^c_c , o^c_e }

Fig. 20. Multimedia playback with skip user interrupt.

the user specified to skip to. This time, p_{ui} acts upon n_{ui} and changes n_1 to the number specified by n_{skip}. n_{ui} is then reinitialized to 1 to allow normal playback. The system subsequently plays the media segment specified by the "skip" command and continues to increment from that position.

5) Integrating User Interrupts: The DPN model in Fig. 21 shows the playback with all the five possible user interrupts. n_{ui} is shared between the different user interrupt types. The default modes, such as Restart and Forward, are assigned the function of $n_{\text{ui}} = 1$, ensuring that there is normal playback from the start since the initialized value of all control variables is 1.

Reverse and Skip are assigned the functions of $n_{\text{ui}} = 2$ and $n_{\text{ui}} = 4$ respectively. These two user interrupts are handled by the control place p_{ui}. Freeze, assigned the state of $n_{\text{ui}} = 3$, is handled by the function is_frz.

E. Issues of DPN

The major issue in DPN is net marking ambiguity. There is a potential problem when two places invoke two separate functions at the same marking and these functions operate to give different values to any common variable. The clash arises when trying to determine which function changes the variable. This clash has been partially solved by the sequence of execution of control function. A work-around is to avoid the usage of control functions that may create this ambiguity such as when two places with control functions acting on the same control variable receive tokens at the same time.

Another work-around may be considered for future work. This is to cause control functions to be assigned priorities in execution, allowing sequential execution of control functions in each marking.

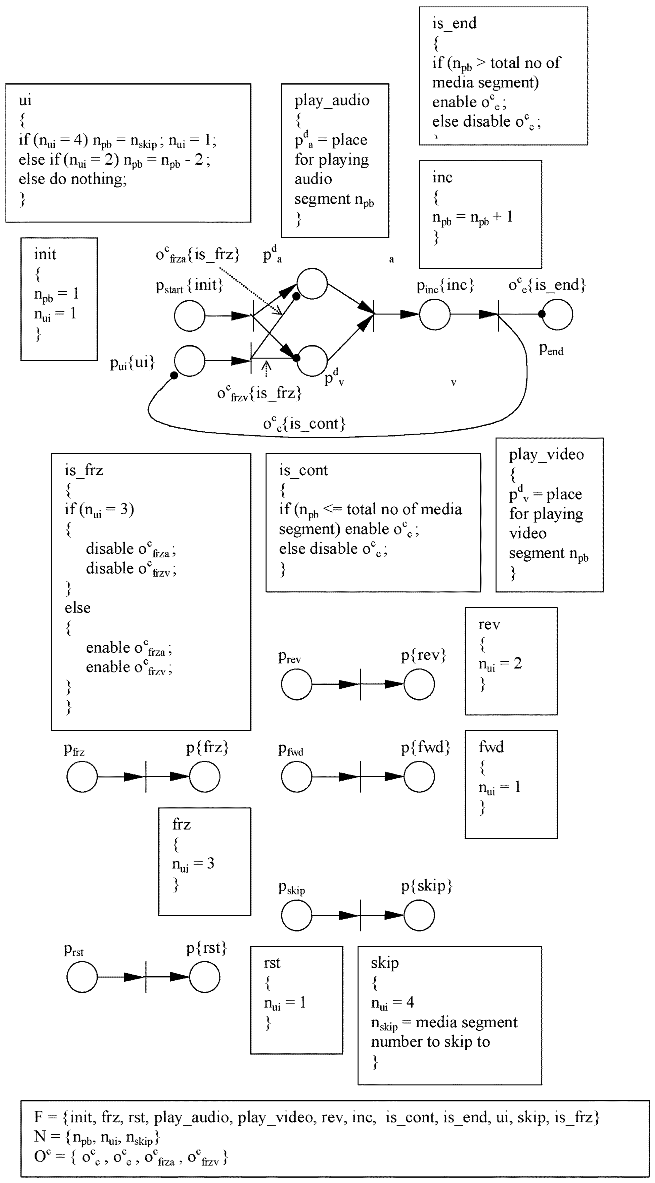

ui { if (n_ui = 4) n_pb = n_skip ; n_ui = 1; else if (n_ui = 2) n_pb = n_pb - 2 ; else do nothing; }
play_audio { p^d_a = place for playing audio segment n_pb }
is_end { if (n_pb > total no of media segment) enable o^c_e ; else disable o^c_e ; }
inc { n_pb = n_pb + 1 }
init { n_pb = 1 n_ui = 1 }
o^c_frza{is_frz}
p^d_a
p_start{init}
p_inc{inc}
o^c_e{is_end}
p_end
p_ui{ui}
p^d_v
o^c_frzv{is_frz}
o^c_c{is_cont}
is_frz { if (n_ui = 3) { disable o^c_frza ; disable o^c_frzv ; } else { enable o^c_frza ; enable o^c_frzv ; } }
is_cont { if (n_pb <= total no of media segment) enable o^c_c ; else disable o^c_c ; }
play_video { p^d_v = place for playing video segment n_pb }
p_rev
p{rev}
rev { n_ui = 2 }
p_frz
p{frz}
p_fwd
p{fwd}
fwd { n_ui = 1 }
frz { n_ui = 3 }
p_skip
p{skip}
p_rst
p{rst}
rst { n_ui = 1 }
skip { n_ui = 4 n_skip = media segment number to skip to }
F = {init, frz, rst, play_audio, play_video, rev, inc, is_cont, is_end, ui, skip, is_frz}
N = {n_pb, n_ui, n_skip}
O^c = { o^c_c , o^c_e , o^c_frza , o^c_frzv }

Fig. 21. Integrating user interrupts.

IV. Design of a Multimedia Orchestration Tool (MOT)

A. Requirements

The implemented multimedia orchestration tool is a networked multimedia client application that works in a distributed multimedia system to deliver the following capabilities.

1) Synchronous Multimedia Playback: The application is to be able to playback video and audio media segments synchronously. This means that video and audio segments are to be played at the right time according to a predetermined order.

2) User Interactivity: The application is to support and respond to a set of pre-defined user interactions. The user interactions that are to be supported are Reverse, Forward, Freeze, Restart, and Skip.

3) Buffering of Data: The application is to allow buffering of data before playback to obtain a reasonable quality-of-service (QoS).

4) Remote Synchronization of Media Playback: The application is to allow remote synchronization from the server. Upon

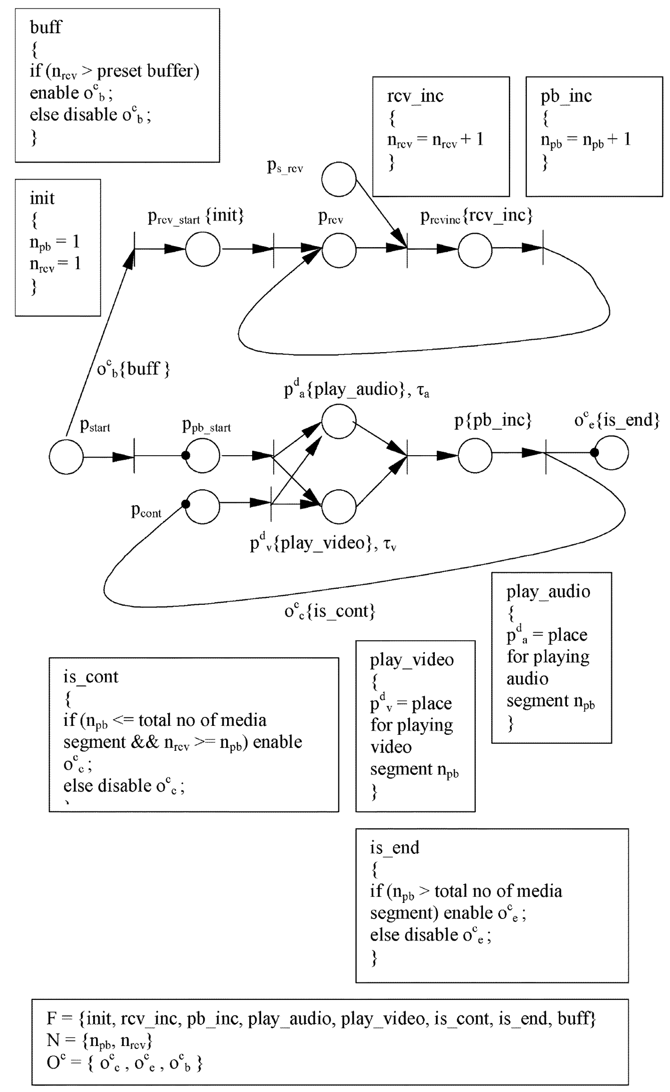

buff { if (n_{rcv} > preset buffer) enable o^c_b ; else disable o^c_b ; }
init { n_{pb} = 1 n_{rcv} = 1 }
rcv_inc { $n_{rcv} = n_{rcv} + 1$ }
pb_inc { $n_{pb} = n_{pb} + 1$ }
p_{s_rcv}
p_{rcv_start} {init}
p_{rcv}
p_{rcvinc} {rcv_inc}
o^c_b{buff}
p^d_a{play_audio}, τ_a
p_{start}
p_{pb_start}
p{pb_inc}
o^c_e{is_end}
p_{cont}
p^d_v{play_video}, τ_v
o^c_c{is_cont}
play_audio { p^d_a = place for playing audio segment n_{pb} }
is_cont { if (n_{pb} <= total no of media segment && n_{rcv} >= n_{pb}) enable o^c_c ; else disable o^c_c ;
play_video { p^d_v = place for playing video segment n_{pb} }
is_end { if (n_{pb} > total no of media segment) enable o^c_e ; else disable o^c_e ; }
F = {init, rcv_inc, pb_inc, play_audio, play_video, is_cont, is_end, buff}
N = {n_{pb}, n_{rcv}}
O^c = { o^c_c , o^c_e , o^c_b }

Fig. 22. DPN showing normal playback.

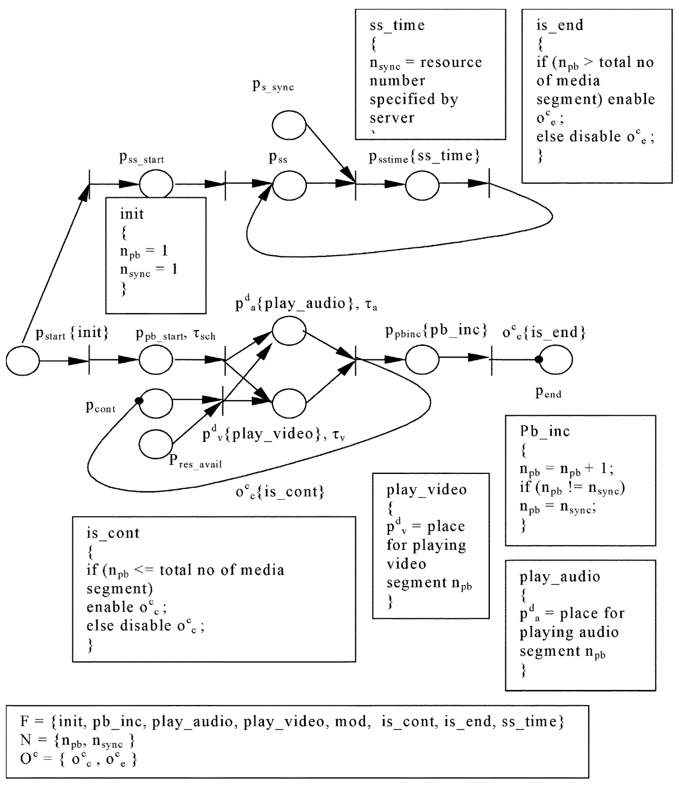

ss_time { n_{sync} = resource number specified by server
is_end { if (n_{pb} > total no of media segment) enable o^c_e ; else disable o^c_e ; }
p_{s_sync}
p_{ss_start}
p_{ss}
p_{sstime}{ss_time}
init { n_{pb} = 1 n_{sync} = 1 }
p^d_a{play_audio}, τ_a
p_{start} {init}
p_{pb_start}, τ_{sch}
p_{pbinc}{pb_inc}
o^c_e{is_end}
p_{end}
p_{cont}
p^d_v{play_video}, τ_v
p_{res_avail}
o^c_c{is_cont}
Pb_inc { $n_{pb} = n_{pb} + 1$; if (n_{pb} != n_{sync}) $n_{pb} = n_{sync}$; }
is_cont { if (n_{pb} <= total no of media segment) enable o^c_c ; else disable o^c_c ; }
play_video { p^d_v = place for playing video segment n_{pb} }
play_audio { p^d_a = place for playing audio segment n_{pb} }
F = {init, pb_inc, play_audio, play_video, mod, is_cont, is_end, ss_time}
N = {n_{pb}, n_{sync} }
O^c = { o^c_c , o^c_e }

Fig. 23. DPN model of server synchronized and scheduled playback.

receiving a synchronization notification from the server, the application will synchronize its playback, holding playback if it has been too fast (ahead of the server) or skipping playback if it is too slow (lagging behind the server). This allows a server to synchronize two or more instances of the client to play media in unison.

5) Remote Scheduling of Media Playback: The application is to support remote scheduling of media playback. The server counterpart of the multimedia orchestration tool may use this feature to start two or more instances of the client at the same time.

The orchestration tool supports server-based synchronization and scheduling of media playback. This is to enable the server to control and conduct the playback of two or more clients such that they play media in unison according to a pre-determined order.

B. Multimedia Orchestration Tool Design

1) Modeling Normal Playback: In normal playback, the orchestration tool detects if the media resource to be played next is available. If it is not available, the system will wait for that resource before continuing with playback.

Before playback is allowed to begin, the client must have received a certain number of files. This is the playback buffer. This ensures that the playback will have adequate resources to consume when it commences.

Fig. 22 is a DPN model of normal playback. The playback is similar to the one shown in Fig. 12. The additional controls within the model are to model buffering and unavailability of resources.

p_{s_rcv} is the place where the client has received a resource from the server. A token is created in p_{s_rcv} each time a resource is received. This in turn causes a token to be created in the control place $p_{\mathrm{rcv\ inc}}$. Function rcv_inc then increments the value of n_{rcv}, the number of files received.

Buffering is modeled by the control arc, o^c_b. o^c_b is disabled when n_{rcv} is less than a preset buffer size. Hence, playback will not commence in the model until the number of resources received is over the preset buffer size.

Unavailability of resources is modeled by including an extra condition to the function controlling the control arc o^c_c. Here, the playback will not continue unless n_{rcv}, the number of resources exceeds or equals the n_{pb}, the number of resources played so far.

2) Playback With User Interrupts: The orchestration tool supports the user interrupts: Freeze, Restart, Skip, Reverse and Forward.

The model shown in Fig. 21 also models the way the orchestration tool services user interrupts.

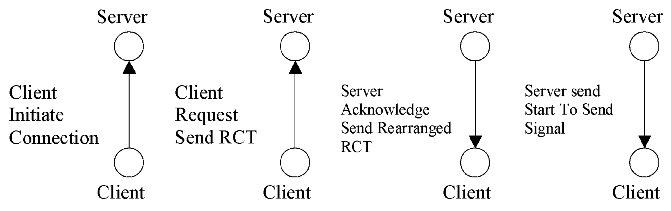

Server
Client Initiate Connection
Client
Server
Client Request Send RCT
Client
Server
Server Acknowledge Send Rearranged RCT
Client
Server
Server send Start To Send Signal
Client

Fig. 24. Handshake and request sequence.

3) Server Synchronized and Scheduled Playback: Fig. 23 shows the DPN model of the server synchronization and scheduling of multimedia playback.

In the beginning, the server provides the client with a time to start. This time is reflected by the time interval τ_{sch} in Fig. 23. When the system is started, a token will be locked in $p_{\mathrm{pb\_start}}$ for the duration of the time interval τ_{sch}. p_{s_sync} is the place where the client receives a synchronization string from the server. The control function ss_time modifies n_{sync} to match the resource number specified by the synchronization string. The function mod from the control place p_{mod} forces the playback resource number to be set to the resource number given by the server hence correcting the playback.

V. Implementation of an Orchestration Tool

The objective of the implementation was to develop a multimedia application to deliver the following capabilities:

— synchronous multimedia playback;
— user interactivity;
— buffering of data;
— remote synchronization of media playback;
— remote scheduling of media playback.

The application is also to allow buffering of data before playback to obtain a reasonable QoS.

A. Overview of the Multimedia Orchestration Tool

The multimedia orchestration tool is a multithreaded Microsoft Windows-based application developed mainly in the C++, using the Microsoft Visual C++ environment. The application utilizes the Microsoft TCP/IP stack within the Win32 environment to communicate with a multimedia resource server. TCP/IP was used because of its widespread use in computer networks today. The application supports audio playback. Video playback was realized by discrete display of JPEG images. The application runs on one main thread and a number of dynamic child threads. This allows the application to support playback of media and accepting user interaction on one thread while receiving media resources on another thread.

B. Implementation

1) Server-Client Handshake and Request: In Fig. 24, the client initiates a TCP/IP connection to the server. The client sends its request as a resource code table (RCT). The Server sends back a rearranged version of the RCT and sends a Start To Send when it is ready to send the data requested.

The Resource Table is headed by the string "RCT". The rest of the table are a number of 3-string entries. The first string entry gives the start time of the resource as a string. The second string entry gives the end time of the resource as a string. The third string consists of the resource code. This resource code corresponds to a multimedia resource that the server has (such as a jpeg or wave file) and identifies that resource to the server for transmission. The resource code and start time will be used to manage playback subsequently. The end time is redundant information and is included for future work.

The Rearranged Resource Table shares the same format as the Resource Table except that it is sent back from the server without the "RCT" header. The server takes the Resource Table and sorts the 3-string entries such that the Rearranged Resource Table has 3-string entries sorted by start time in ascending order from the head of the list to the tail.

The client allows the user to enter the start time, end time, start frame and end frame of the resources being requested. The resource code is generated according to which resource type is selected by the user.

2) Buffering: Before playback is allowed to begin, the client must have received a certain number of files. This is the playback buffer. This ensures that the playback will have adequate resources to consume when it commences.

3) Normal Playback: In normal playback, the orchestration tool detects if the media resource to be played next is available. If it is not available, the system will wait for that resource before continuing with playback.

4) Playback With User Interrupts: The orchestration tool supports the following user interrupts: Freeze, Restart, Skip, Reverse, Forward.

The procedure for notifying a server of a user interrupt is as follows.

a) Pause playback.
b) Retrieve current state.
c) Generate user interrupt resource table string list.
d) Start new socket on new thread.
e) Wait for Start To Send from the server.
f) When the server has signaled Start To Send, close old socket and old thread.

5) Server Synchronized and Scheduled Playback: The procedure of server scheduling and synchronization is as follows.

a) Client sends a request as usual.
b) Upon filling its initial buffer, it sends a string "RSS" to the server.
c) The server responds by sending back a stringlist containing the server time and the offset to start.
d) The client determines the time to start using the difference between the server time and client time and adds the offset and starts a timer.
e) When the timer is up, the playback starts.
f) The server will send periodic strings of the relative time elapsed.
g) The client will translate this code to update and correct its playback accordingly.

VI. Conclusions and Future Work

The DPNt is a model that brings the power of programmability to Petri Nets and hence solves the problem of net complexity when it comes to modeling synchronous playback of multimedia. DPN has also introduced new solutions to the modeling of asynchrony with the flexibility of control functions. The major issue of DPN is its susceptibility to clash in the case that two control functions try to change a common variable concurrently.

The multimedia orchestration tool implemented was able to utilize DPN in the design and modeling of its different aspects, notably the handling of user interaction and server-based synchronization. The multimedia orchestration tool is a typical representation of an application that exists in a distributed multimedia system environment.

Future work may be done on prioritizing DPN's control functions to avoid the clash mentioned above. In addition, to help users generate efficiently presentation schedules for multimedia presentations, some DPN-based authorware, toolkits or middleware can be developed so that DPN building blocks or reusable objects are provided. This will also improve the quality of design as the correctness of such building blocks or reusable objects should have been verified extensively.

References

[1] F. Furtek, "A new approach to Petri Nets," *MIT Project MAC*, Apr. 1975.

[2] K. Jensen, *Colored Petrinets: Basic Concepts, Analysis Methods and Practical Use*, 2nd ed. New York: Springer, 1997, vol. 1, pp. 2–9.

[3] T. C. D. Little and A. Ghafoor, "Synchronization and storage models for multimedia objects," *IEEE J. Select. Areas Commun.*, vol. 4, pp. 413–427, Apr. 1990.

[4] 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.

[5] S.-U. Guan, H.-Y. Yu, and J.-S. Yang, "A prioritized petri net model and its application in distributed multimedia systems," *IEEE Trans. Comput.*, vol. 47, no. 4, Apr. 1998.

[6] S.-U. Guan and S.-S. Lim, EP-Net: A Synchronization Model for Authoring Interactive Multimedia Applications, 1999.

[7] K. Prabhat, B. Andleigh, and T. Kiran, *Multimedia Systems Design*. Englewood Cliffs, NJ: Prentice-Hall, 1996, pp. 421–444.

[8] C.-M. Huang, C. Wang, and C.-Y. Kuo, "A master-medium-based interactive synchronization control scheme for distributed multimedia systems," in *Euromicro Conf. Proc.*, vol. 2, 1998, pp. 506–513.

[9] J. J. P. Tsai, Y. Bi, S. J. H. Yang, and R. A. W. Smith, *Distributed Real-Time Systems*. New York: Wiley, 1996, ch. 11, pp. 247–275.

[10] P. Lougher, "The design of a storage server for continuous media," *Comput. J.*, vol. 36, 1993.

[11] S. M. Chung, *Multimedia Information Storage And Management*. Norwell, MA: Kluwer, 1996, pp. 303–411.

[12] H. Thimm and W. Klas, "Managing adaptive presentation executions in distributed multimedia database system," in *Proc. 1996 Int. Workshop On Multimedia Database Management Systems*, Aug. 1996, pp. 152–167.

[13] G. Bruno, *Model-Based Software Engineering*. London, U.K.: Chapman & Hall, 1995, pp. 63–101.

[14] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[15] B. Prabhakaran and S. V. Raghavan, "Synchronization models for multimedia presentation with user participation," *ACM Multimedia Proc.*, pp. 157–166, Aug. 1993.

[16] K. Yoon and P. B. Berra, "TOPCN: Interactive temporal model for interactive multimedia documents," in *Proc. Int. Workshop on Multimedia Database Management Systems*, Aug. 1998, pp. 136–144.

[17] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modeling with Generalized Stochastic Petri Nets*. New York: Wiley, 1996, ch. 3, pp. 49–68.

[18] A. F. Tanenbaum, *Computer Network*. Englewood Cliffs, NJ: Prentice-Hall, 1996, pp. 219–239.

[19] N. U. Qazi, M. Woo, and A. Ghafoor, "A synchronization and communication model for distributed multimedia objects," in *Proc. 1st ACM Int. Conf. Multimedia*, Aug. 1993, pp. 147–155.

[20] M. Diaz and P. Senac, "Time stream petri nets a model for multimedia streams synchronization," in *Proc. 1st Int. Conf. Multimedia Modeling*, 1993, pp. 257–273.

Roy Tan received the B.Eng. degree in electrical engineering from the National University of Singapore.

Since 2000, he has been an Engineer with Agilent Technologies, where he currently develops isolation products for Agilent's Semiconductor Products Group. His current research interests include OS design, cryptography, and microprocessor architecture.

Sheng-Uei Guan received the M.Sc. and Ph.D. degrees from the University of North Carolina at Chapel Hill.

He is currently with the Electrical and Computer Engineering Department, National University of Singapore. He has worked in a prestigious R&D organization for several years, serving as a Design Engineer, Project Leader, and Manager. He has also served as a member on the R.O.C. Information and Communication National Standard Draft Committee. After leaving the industry, he joined Yuan-Ze University in Taiwan for three and half years. He served as a Deputy Director for the Computing Center and as Chairman for the Department of Information and Communication Technology. Later, he joined the Department of Computer Science and Computer Engineering, La Trobe University, where he helped to create a new Multimedia Systems stream.