Self-Modifiable Color Petri Nets for Modeling User Manipulation and Network Event Handling

Sheng-Uei Guan and Wei Liu

Abstract—A Self-Modifiable Color Petri Net (SMCPN) which has multimedia synchronization capability and the ability to model user manipulation and network event (i.e., network congestion, etc.) handling is proposed in this paper. In SMCPN, there are two types of tokens: resource tokens representing resources to be presented and color tokens with two subtypes: one associated with some commands to modify the net mechanism in operation, another associated with a number to decide iteration times. Also introduced is a new type of resource token, named reverse token, that moves in the opposite direction of arcs. When user manipulation/network event occurs, color tokens associated with the corresponding interrupt handling commands will be injected into places that contain resource tokens. These commands are then executed to handle the user manipulation/network event. SMCPN has the desired general programmability in the following sense: 1) It allows handling of user manipulations or prespecified events at any time while keeping the Petri net design simple and easy. 2) It allows the user to customize event handling beforehand. This means the system being modeled can handle not only commonly seen user interrupts (e.g., skip, reverse, freeze), the user is free to define new operations, including network event handling. 3) It has the power to simulate self-modifying protocols. A simulator has been built to demonstrate the feasibility of SMCPN.

Index Terms—Self-Modifiable Color Petri Nets (SMCPN), color tokens, multimedia synchronization, self-modifying protocols.

1 Introduction

Multiple streams of data from different sources may become out-of-sync when the streams are being sent and displayed across the network. This can be annoying or even unacceptable for entertainment or commercial purposes. So, synchronization is an important problem in distributed systems.

Synchronization of a multimedia system can be divided into two aspects: spatial composition and temporal composition. Spatial composition is needed in intermedia synchronization, which is to maintain the spatial relationship between media, e.g., image and text. Temporal composition is needed both in intermedia and intramedia composition, which is to maintain the temporal relationship within the media.

User manipulation (or interrupt) requirements describe what user input and output an application needs to support. For example, a karaoke application is expected to support a number of interrupts, such as speed scaling, muting the sound of the singer, etc. It is an important factor to evaluate a model to see if it can handle user interrupts.

Since Petri net was proposed, it has been used in domains such as design and simulation of control systems, communication protocols, and manufacturing processes [15]. With many extended Petri net models proposed, the application of Petri nets has been extended to multimedia systems. The focus of the research flows from the synchronization of multimedia without user interactions to interactions in distributed environments [2], [3, [4], [5], [6], [8], [9], [11], [12]. Some of the models are: Object Composite Petri Net (OCPN) [14], Dynamic Timed Petri Net (DTPN) [1], Prioritized Petri Net (P-Net) [11], and Enhanced Prioritized Petri Net (EP-Net) [12].

An extended Petri net model—Self-Modifiable Color Petri Net (SMCPN) is proposed in this paper. This model can be used to handle user manipulations and network events such as network congestion. None of the extended Petri nets mentioned above have demonstrated controllability and programmability as SMCPN has offered. They do not have the ability to reduce modeling size, improve design flexibility, model playback on-the-fly, or simulate real-time adaptive applications.

The paper is organized as follows: Section 2 gives an overview on some major extended Petri nets used in multimedia applications. Section 3 presents the definitions of SMCPN with an example. Section 4 explains how to achieve synchronous control of user interrupts/network events based on SMCPN. Section 5 introduces an SMCPN simulator. Section 6 concludes the paper and proposes future work. The Appendix contains more examples of SMCPN handling user interrupts.

2 Related Work

2.1 Petri Nets

Petri net is a graphical tool for the formal description of systems whose dynamics are characterized by concurrency, synchronization, and mutual exclusion, which are typical features of distributed environment. The formal definition of Petri nets is a four-tuple (P, T, I, O) [7], where P is a set of places corresponding to states, T is a set of transitions which correspond to the rules of firing or nonfiring. I and O are the input and output functions that can change the states. The dynamic performance of a Petri net is controlled by the

• *The authors are with Department of Electrical and Computer Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260. E-mail: {eleguans, eleliuw}@nus.sed.sg.*

Manuscript received 30 Jan. 2002; accepted 28 Mar. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 115795.

firing rule. A transition will fire if all its input places contain at least one token.

Several extended Petri net models have been proposed to extend its application domains. In this section, we describe some of these models.

2.1.1 Colored Petri Net (CPN)

CPN [16] overcomes the weakness of Petri net in describing complex systems. By introducing color sets, CPN can achieve compact representation by equipping each token with an attached data value, i.e., token color. CPN, especially hierarchical CPN, has mainly been used in modeling and analyzing large systems such as VLSI chip design and electrical funds transfer [16]. CPN is not meant for modeling the target applications of the SMCPN model.

2.1.2 Object Composition Petri Net (OCPN)

The OCPN [14] model proposed by Little and Ghafoor is an enhanced version of Timed Petri nets. It introduces the value of time as duration to the conventional Petri nets. OCPN can model temporal relations between media data in multimedia presentation. This is the most important contribution of this model. This model cannot handle user interrupt and distributed environment.

2.1.3 Dynamic Timed Petri Net (DTPN)

The DTPN [1] approach proposed by Prabhakaran and Raghavan is an extension of OCPN that has the power to deal with user interrupt. DTPN extends OCPN to model typical user interrupts such as skip, reverse, freeze, restart, and scale up/down the speed of a presentation. These are achieved by allowing preempting of the Petri net execution sequence and modification of the playback duration in the places. When a place is preempted, the modification of its execution duration will have four different types: temporary, deference, termination, and permanent. Although DTPN can handle some user interrupts, it still cannot deal with the premature/late arriving tokens. The definitions of premature/late arriving tokens are given in the paper [11].

2.1.4 Prioritized Petri Net (P-Net) and Distributed Object Composition Petri Net (DOCPN)

To overcome the limitation of the models introduced above, Guan et al. have proposed DOCPN [11]. DOCPN inherits the characteristics of Petri net and uses the OCPN synchronous methods to synchronize among intermedia objects. The main contributions of this model are: 1) It extends OCPN to the distributed environment using a global clock, and enables user interrupt control into OCPN. 2) A new mechanism, named prioritized Petri net (P-net), is introduced, the arrival of a priority input event (e.g., user interrupt) can fire a transition without waiting for other nonpriority events. The main concern here is what happens if a token arrives at a nonpriority input place after the transition has been forced to fire by an earlier priority input event. This motivates the design of EP-nets [12].

2.1.5 Enhanced Prioritized Petri Net (EP-Net)

Proposed by Guan and Lim, Enhanced Prioritized Petri Net (EP-net) [12] is an upgraded version of P-net. EP-net uses a mechanism known as Premature/Late Arriving Token Handler (PLATH) to handle late and/or premature tokens (locked tokens forced to unlock). Moreover, EP-net introduces mechanisms such as PLATH and dynamic arcs that can be associated with sets of program statements to handle user interrupts; this improves the flexibility of designing interactive systems. Although EP-net is more powerful than P-net, it needs to duplicate a reversed Petri net every time a reverse operation is called, which means that the model size and effort will increase tremendously when the network size increases. In order to overcome this weakness, SMCPN comes to rescue.

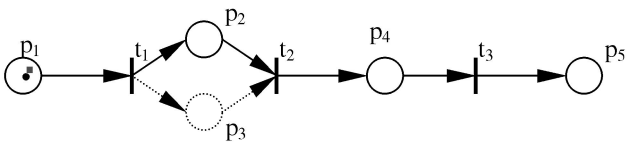


Fig. 1. SMCPN: An example. • Color token c: Modify the net structure by adding a place p_3 and two arcs $t_1 = p_3$, $p_3 - t_2$.

2.1.6 Synchronized Multimedia Integration Language (SMIL)

SMIL [23]—a W3C standard, allows integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can: 1) describe the temporal behavior of the presentation, 2) describe the layout of the presentation on a screen, 3) associate hyperlinks with media objects. The difference between SMIL and SMCPN is that SMCPN can deal with the indeterministic part of temporal schedule, i.e., temporal schedule changeable and programmable due to (new types of) user or network interrupts or manipulations at runtime, while SMIL focus on the layout and prescription of temporal schedule due to fixed types of user interrupts. SMCPN allows runtime change of the temporal model itself with the changes preprogrammed to handle any type of user or network interrupts customizable by the user or network manager.

3 Self-Modifiable Color Petri Net

In this section, we introduce the general concepts and definitions of Self-Modifiable Color Petri Nets (SMCPN). SMCPN can model large-scale synchronization, supporting user interrupts in distributed environments and handling network events such as congestion.

3.1 Definitions

In SMCPN, the tokens are divided into two types: color tokens and resource tokens. Furthermore, there are two types of color tokens: one associated with commands to accomplish control, another associated with a number that indicates the number of iterations needed to play. There are also two types of resource tokens used to play back resources: a forward one that moves in the same direction with arcs and a reverse one that moves in the opposite direction with arcs. In Fig. 1, an example of SMCPN is illustrated. The mechanisms drawn in dotted lines are created after the control commands associated with the color token *c* are executed. When the color token *c* appears

TABLE 1
List of Commands

No	Mechanisms	Commands	Actions
1	Arc →	Create arc*	An arc is created
2		Delete arc*	An arc is deleted
3		Create discard arc	An arc to discard some resource is created
4	Place ○	Create place*	A place is created
5		Delete place*	A place is deleted
6		Replace place	A place is replaced
7		Change duration*	Change the duration associated with a place
8	Transition	Enable transition	A transition is able to fire when the conditions to fire are met
9	|	Disable transition	A transition is not able to fire regardless whether the conditions to fire the transition are fulfilled
10		Create transition*	A transition is created
11		Delete transition*	A transition is deleted
12	Token:	Lock token	To lock a token
13		Unlock token	To unlock a token
14		Reverse	Change the direction of the resource token
15	Color token ▪ Forward token	Pause token	To stop counting down if a place is associated with a duration or stop a transition from firing if the place is associated with no duration
16	•	Resume token	To resume a token and start counting down from the time it has been paused
17	Reverse token	Create token* (Color, normal)	To create a token to the indicated place with no condition
18	°	Delete token* (Color, normal)	To remove a token
19		Disable token	Let a token run without any effect. Given an audio signal as an example, let it go through the net with no sound. Useful for karaoke functions
20	Number	Increase value*	Increase the value of the number associated with a color token
21		Decrease value*	Decrease the value of the number associated with a color token

in the place p_1, the commands associated with it will be executed. In SMCPN, we have some primitive commands that can modify the net structure such as to create a place, disable a transition, etc. We can combine several of these basic commands in a special sequence to handle different user interrupts/network events.

There are two approaches to represent modification commands: One is using one color token to represent one type of (user) interrupt, thus a color token represents a combination of several basic commands. Another is using one color token to represent one basic command and the color of a token also indicates the priority of a command (i.e., which command will be executed first). Because the former approach is closer to reality, we use it in this paper. In the example illustrated by Fig. 1, the color token *c* is associated with five basic commands: disable transition t_1, create a presentation place p_3, create an arc from transition t_1 to place p_3, create an arc from place p_3 to transition t_2, enable transition t_1. The part drawn in dotted lines do not exist at the beginning, then the color token *c* is injected into p_1 due to some interrupts/events. The implementation of the resource token in place p_1 will be paused, then the modification commands associated with *c* is executed. The place and arcs drawn in dotted lines are created successively. The mechanism of SMCPN has been shown. In the following, we will introduce the definitions of SMCPN.

Definition 1 (SMCPN). *SMCPN is a seven-tuple* $\mathrm{S} = \{\mathrm{P}, \mathrm{T}, \mathrm{A}, \mathrm{D}, \mathrm{U}, \mathrm{C}, \mathrm{M}\}$, *where* $\mathrm{P} \cap \mathrm{T} \neq \emptyset$.

$\mathrm{P} = \{\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3, \ldots, \mathrm{p}_\mathrm{m}\}$ is a finite set of places representing states in a multimedia system or processes in network transmission, where $\mathrm{m} > 0$.

$\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3, \ldots, \mathrm{t}_\mathrm{n}\}$ is a finite set of transitions representing synchronization points in a multimedia system/network transmission process, where $\mathrm{n} > 0$.

$\mathrm{A} : \{\mathrm{PXT}\} \cup \{\mathrm{TXP}\}$ is a finite set of arcs representing the flow relation.

$\mathrm{D} : \mathrm{P} \rightarrow \mathrm{R}^+$ is a mapping from the sets of presentation places to the nonnegative real numbers, representing the presentation duration of the resources concerned/transmission speed of network.

$\mathrm{U} = \{\mathrm{u}_1, \mathrm{u}_2, \mathrm{u}_3, \ldots, \mathrm{u}_\mathrm{l}\}$ is a finite set of colors; each color could represent one type of user interrupt/network event, where $\mathrm{l} > 0$. $\mathrm{C} = \{\mathrm{c}_1, \mathrm{c}_2, \mathrm{c}_3, \ldots, \mathrm{c}_\mathrm{k}\}$ is a finite set of commands (as defined in Table 1), where $\mathrm{k} > 0$. (e.g., $\mathrm{c}_1 =$ "add a place," $\mathrm{c}_2 =$ "delete a place"). By using these basic

commands, a user can map each type of user/network operation into a color token.

$M : P \rightarrow \{I_r^+, I_c^+\}$ (I_r—counts the number of resource tokens; I_c—counts the number of color tokens). $I_{c/r}^+ = \{0, 1, 2, \ldots\}$ is a mapping from the set of places to the set of integer numbers, representing a marking of a net.

The list of commands shown in Table 1 is sufficient for any type of change to a Petri net because the basic modifications to Petri nets have been included in our list (the commands labeled with *).

In SMCPN, when a color token is injected into a place, the commands associated with it will be triggered. Different color tokens can be associated with different user interrupts/network events. The main job of a designer is to install the basic commands set and design SMCPN based on special requirements from different user interrupts/network events. A more flexible way is to let a user design an SMCPN itself using these basic commands sets. This makes the usage of SMCPN more elastic. We illustrate this by some solid examples in Section 6.

3.2 Enabling and Firing Rules of SMCPN

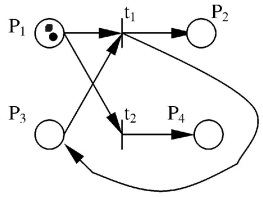
A transition in a Petri net is enabled if all its input places are comprised of a certain number of tokens greater than or equal to the number of each respective place's arcs to the transition and no interrupt disables it. If the conditions mentioned are met, the transition fires and token(s) are removed from each of its input places and token(s) are created at each of its output places. The transition fires instantly if each of its input places is not associated with any duration or contains any unlocked token. In the case when a place is associated with duration, the place remains in the active state for an interval specified by the duration *D* after receiving a token. During this period, the token is locked. Upon the cessation of the duration *D*, the token becomes unlocked. For SMCPN, if no user disables a transition, it will obey this enabling rule of Petri net.

By introducing the novel mechanisms, color token and the primitive control commands, SMCPN supports interactive/distributed multimedia applications and can manipulate the flow of the Petri net modeled. Besides supporting the conventional rules of Petri nets, some new SMCPN rules are needed.

Enabling Rules:

1. Create color tokens for control: When a user interrupt/network event occurs, the color tokens corresponding to this interrupt will be created and injected into the places that contain resource tokens. The commands associated with each color token will be executed in order.
2. Once a color token is injected into a place, the execution of the resource token(s) in this place will be paused. At the same time, the system will not be able to take further interrupts.
3. When a color token is created, the commands associated with it will be executed. The network structure will be modified to satisfy the demands of the corresponding interrupt.

Firing Rule: When all commands associated with a color token have been executed, this color token will be deleted.



Commands list of color token c (●):
1. enable the transition t_1; 2. unlock the token in place p_1
3. enable the transition t_2; 4. disable the transitions t_1 and t_2
p_1 represents a presentation place to start the zeroing test
p_2 represents p_3 has a token; p_3 represents the place to be tested
p_4 represents p_3 has no token

Fig. 2. Simulated zero-token test by SMCPN.

At the same time, the normal resources tokens will be resumed. In particular, the color token associated with an index *N* will be deleted when the index number has been decremented to zero. Then, the enabled transition will fire according to the basic Petri net firing rules.

Finishing: When a color token finishes its operation, all the modifications made for the user interrupt/network event will be removed by default, i.e., the net structure will be restored to the configuration before the user interrupt/network event occurs. Thus, the modifications from an interrupt will not affect the next presentation. If the user chooses to preserve the modification, then the changed net structure will stay.

With the introduction of these new mechanisms, SMCPN will not possess the safeness and conservation property. Although these properties are sacrificed, SMCPN has gained more expressive power in the aspect of general programmability and scalability when handling user or network events. For example, SMCPN has functions to let a user design his own operations. Comparing with the sacrifice in verifiability, benefits from the new mechanisms justify the cost paid.

3.3 Expressive Power of SMCPN

Agerwala [13] and others have shown that an extended Petri Net model with the ability to test a place for zero token can simulate a Turing machine. In the following, we prove that SMCPN has the same expressive power as Turing Machine, i.e., it has the ability to test a place for zero token.

Theorem. *SMCPN has the power to model a Turing Machine.*

Consider p_3 representing a place to be tested, p_1 representing a place to start the zeroing test, p_4 representing p_3 has no token, and p_2 representing p_3 has a token as shown in Fig. 2.

Now, let's see how Fig. 2 can simulate the zero-token test; the initial marking is shown in Fig. 2. At the beginning, transitions t_1 and t_2 are disabled. When a token arrives at place p_1, the resource token in p_1 is paused and the zeroing test starts off. The control commands of color token c will be executed according to the given order. Here, we must pay attention to the execution sequence, transition t_1 must be enabled before transition t_2 and, otherwise, it will produce error results. Assume p_3 contains no token, first commands 1 and 2 are executed, transition t_1 is enabled, and the resource

token in p_1 is unlocked because there is no token in place p_3, so t_1 couldn't fire, no token will be created in place p_2. Then, command 3 is executed, transition t_2 is enabled. Because the input place of transition t_2 contains a resource token, t_2 will fire, and the resource token in place p_1 will move to place p_4. At last, command 4 is executed, transitions t_1 and t_2 are disabled. As a result, there is no token in place p_2 and there is a token in place p_4, indicating that p_3 contains no token. Repeat the whole presentation again, but this time p_3 contains a token. When transition t_1 is enabled, for all of its input places that contain tokens, t_1 will fire and the tokens in places p_1 and p_3 will be removed and a token will appear in place p_2. After this command is finished, t_2 will be enabled, but now there is no token in place p_1 and t_2 will not fire. So, no token will appear in place p_4. Thus, we have proven that SMCPN can model the Turing Machines. In order to preserve the token in place p_3, we set an arc from t_1 to p_3.

4 Modeling Iterative Multimedia, User Interrupts, and Network Events

A color token has the authority to manipulate existing mechanisms or generate new mechanisms. This makes SMCPN powerful in handling user interrupts and network events. Whenever a user interrupt/network event occurs, color tokens will be created and injected into places that contain resource token(s). The commands associated with each color token will be executed and the interrupt is implemented. Using the traditional Petri nets to model a multimedia system, every segment needs to be represented in the model. The model will be huge if there are many segments to be executed. Hence, in SMCPN, a color token with iteration index is introduced to tackle this problem. Besides these, the embedded self-modification function makes SMCPN so powerful that it can be used in modeling self-modifying protocols. In the following, we give some examples to show how SMCPN handles iterative multimedia, user interrupts/network events such as reverse, network congestion, and how to use SMCPN to design self-modifying protocols. For the remaining user interrupts like skip, please refer to the Appendix.

4.1 Modeling Streaming Multimedia Iteration

Consider a media presentation system of an indeterminate number N of similar media resources. Using iteration, presentation of this system is easy to handle. This system is modeled as shown in Fig. 3a.

Upon starting as shown in Fig. 3b, a color token that has been initialized with a value N will be injected into the place p_{dec}. When $N > 0$, arc o_e^c will be disabled and o_c^c will be enabled. During the next transition, as shown in Fig. 3c, the places for audio and video will play the first segment of each resource. In Fig. 3d, a resource token is now in p_{dec}. When a resource token arrives in this place, the value of N will be decremented by one. Assuming that N is greater than 0, o_c^c remains enabled and o_e^c remains disabled. Thus, a resource token will go back to p_{start} in the next instantiation. In Fig. 3e, it is easy to see that p_{start} allows the presentation to continue. The places p_v^d and p_a^d will play the next video and audio media segments $n+1$. The presentation goes on until N goes down to 0. In this case, arc o_e^c will be enabled and o_c^c will be disabled, thus ending the presentation shown in Fig. 3f. At the same time, the color token will self-delete.

4.2 Reverse Operation

The reverse operation is similar to the forward operation, only the flowing direction of tokens is opposite. In SMCPN, the proposal of reverse tokens makes it is easy to handle "reverse" interrupt. Sometimes, the "reverse" operation can also be combined with the "speed scaling" operation to form a "fast reverse" operation.

The handling of "reverse" interrupt using SMCPN is illustrated in Fig. 4.

The implementation of "reverse" can be demonstrated by the above figures. In Fig. 4a, the pointer at p_5 represents where a "reverse" user interrupt occurs. When a user interrupt occurs, a color token (the triangle type token) corresponding to the reverse operation will be created and injected into the place p_5 where there is a token in it, as shown in Fig. 4b. The presentation of token a in p_5 will be paused at the same time when the color token appears in p_5. In Fig. 4c, the commands associated with the color token will be executed, the forward token a in p_5 is changed into a reverse token. In Fig. 4d, the color token is deleted and the net is resumed, the token is moved to the opposite direction of the arcs from p_5 to p_4. In Fig. 4e, the implementation of reverse operation is finished, the resource token a arrives at place p_1. At last, the reverse token is removed and the forward token is restored, as shown in Fig. 4f.

A "reverse" interrupt applied to an iterated presentation also can be handled by a color token tagged with a number, as shown in Fig. 4g. When a "reverse" interrupt occurs, a color token (the solid token) associated with a number N (which represents the number of segments to be reversed) will be injected into p_{dec}. When the color token is injected into the place, the resource token will be changed to a reverse one and moved to the opposite direction of the arcs with the color token(s). The number N associated with the color token(s) will be decreased by one each time the reverse token comes into place p_{dec}. If N is greater than zero, reverse will be continued. When N counts down to zero, the reverse operation will end. The reversed resource token will be changed to a forward token again and the color token will be deleted.

4.3 Handling Network Congestion

Interactive multimedia communicaton basically means two-way processing of multimedia data between the users and a variety of sources and destinations. Sometimes network congestion arises when too much data are transmitted at the same time.

Here, we use SMCPN to model multimedia data transmission in networks. When network congestion occurs, there are options to handle this event under SMCPN. Users are allowed to choose an appropriate handling method according to the importance of data before transmission starts. If the data is important—no packet should be lost, users can choose to scale down the transmission speed, as shown in Fig. 5. When SMCPN is used in modeling data transmission, the duration associated with a place will represent the transmission speed. As shown in Fig. 5a, when network congestion occurs at p_3&p_4,

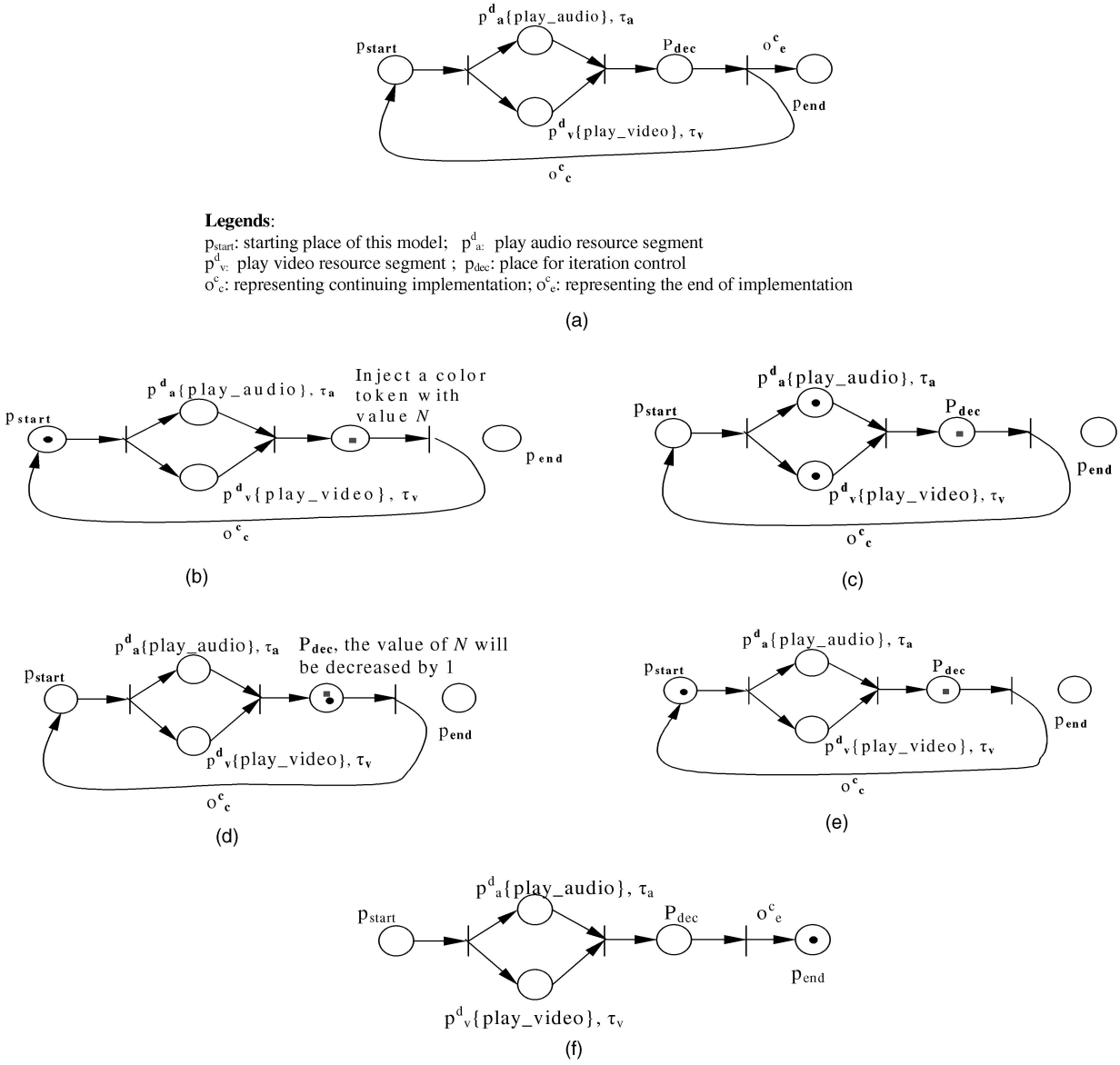


Fig. 3. Presentation of streaming media in SMCPN. (a) SMCPN model of streaming media presentation. (b) Zeroth instantiation—streaming media presentation. (c) First instantiation—streaming media presentation. (d) Second instantiation—streaming media presentation. (e) Third instantiation—streaming media presentation. (f) Final instantiation—streaming media presentation.

two color tokens associated with the change duration command will be injected into p_3&p_4. Then, the commands will be executed, the durations of places p_3&p_4 are changed, as shown in Fig. 5b. As the congestion condition could be worse than expected, the system will do a better job by exploring the speed by decreasing it successively until the congestion is relieved. When a suitable speed is found, the congestion color token will self-delete and data transmission will continue with the new transmission speed, as shown in Fig. 5c.

Alternatively, when data is not so important such as video segments—losing some packets will not affect the quality seriously, users can choose to discard some packets to relieve congestion. Before data transmission, users can set the handling method to packet removal. Thus, when network congestion occurs, some packets will be lost to relieve traffic jam, as shown in Fig. 6. And, the duration associated with such places can be adjusted to specify the duration for packet removal.

4.4 Self-Modify Protocol Design Using SMCPN

In the following, we show how SMCPN can be used to model self-modifying protocol execution. First, we give a self-modifying protocol example. And then, we show the SMCPN to model it. Self-Modifying Protocol (SMP) is a set of instructions, rules, or conventions that can be changed by the systems that communicate with the help of that protocol.

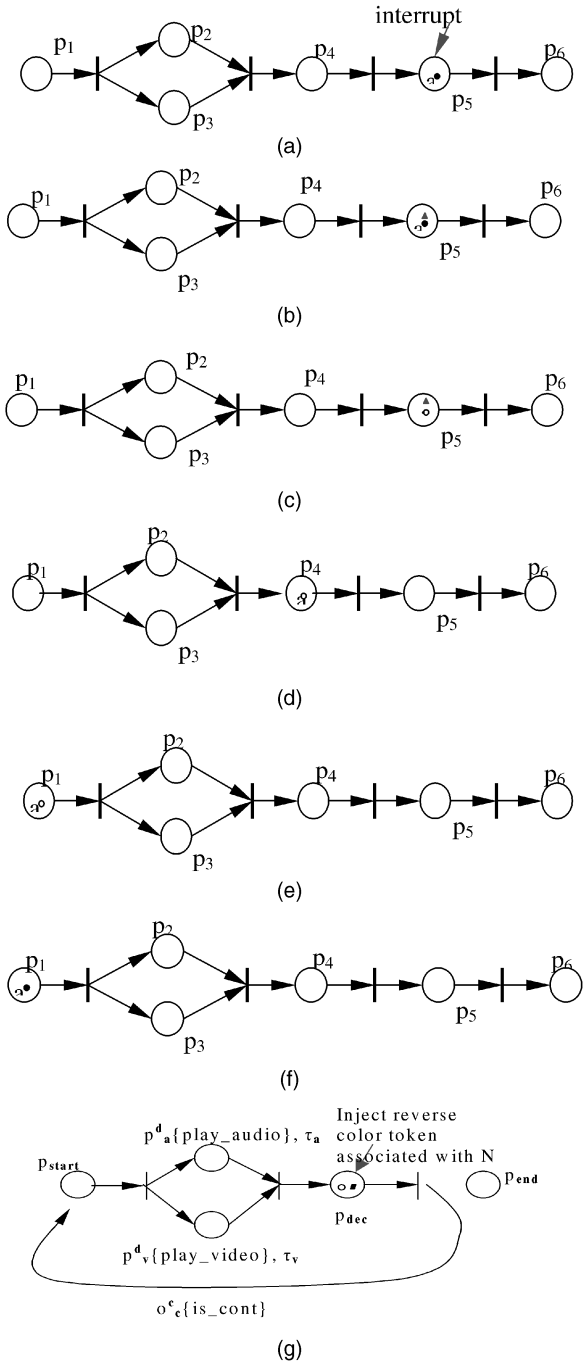


Fig. 4. Handling a reverse interrupt in SMCPN. (a) A "reverse" interrupt occurs. (b) A color token is created and injected into p_5. (c) The command being executed. (d) The modified net is implementing "reverse." (e) The implementation finished. (f) The network restored. (g) Multimedia presentation with a reverse user interrupt in SMCPN.

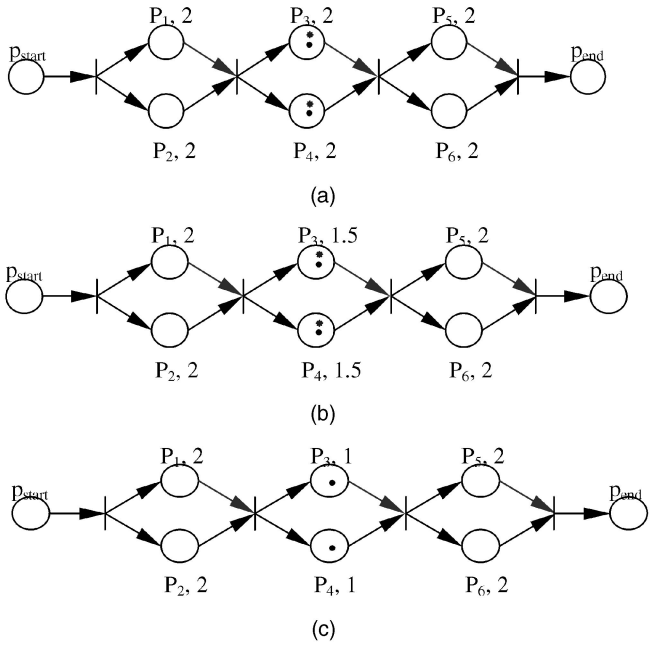


Fig. 5. Changing transmission speed to handle network congestion. (a) Network congestion occurs at $p_3\&p_4$. (b) $p_3\&p_4$ with new transmission speeds. (c) The congestion color token self-deletes and transmission continues.

4.4.1 Self-Modifying Alternating Bit Protocol

The original Alternating Bit Protocol (ABP) can be defined as follows:

- The sender sends its data messages, one by one, to the receiver, but, after sending each data message, it must wait for an acknowledgment before sending the next data message.
- Whenever the receiver receives a data message, it should be able to detect whether it has received an identical copy of this message earlier. For this reason, the value of some bit in the sender is attached to each data message sent. So long as a data message is being resent, the value of this bit remains fixed, but, whenever a new data message is about to be sent, the value of this bit is altered (hence the name "alternating bit").

ABP is designed for communication without error, which is the common scenario during data transmission. Occasionally, network congestion or error can occur and then ABP is not sufficient. SMP (e.g., Self-Modifying ABP) has been proposed in [17] so that it is lightweight during normal communication; however, it is capable of self-modification to deal with exceptions when network events arise.

An SMP example has been presented in [17], as shown in Fig. 7. When the sender does not receive any acknowledgment for some time or the receiver receives a corrupted message (Err), some new transitions will be added to the sender and some new states (N_1, N_2) and transitions will be added to the receiver. Self-modification is introduced upon serious network events. The protocol change created is meant to deal with the event raised.

Using SMCPN to model ABP self-modification, an incoming event (e.g., message received, message error, or loss) will trigger an interrupt upon which a corresponding color token is created based on the type of events occurred or messages received. The color token created will then be injected into the place where it has a normal token. Different events/messages received will lead to different implementation, as shown in Fig. 8. We give a detailed description in the following.

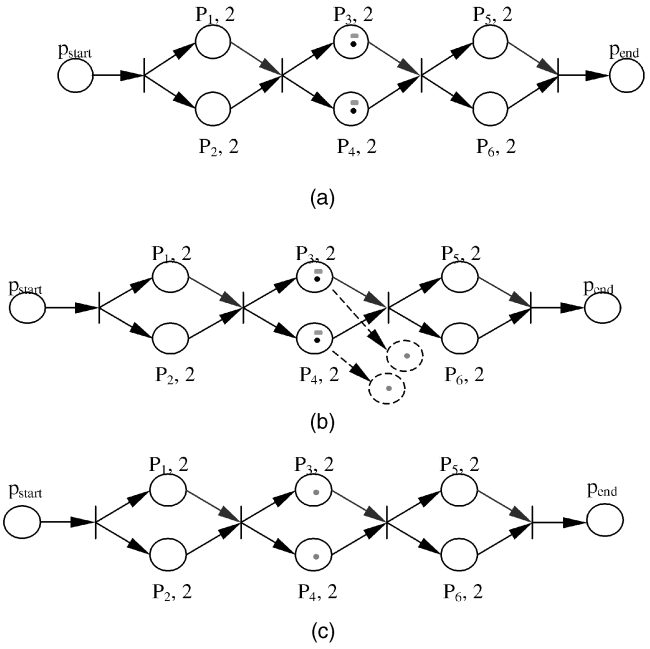


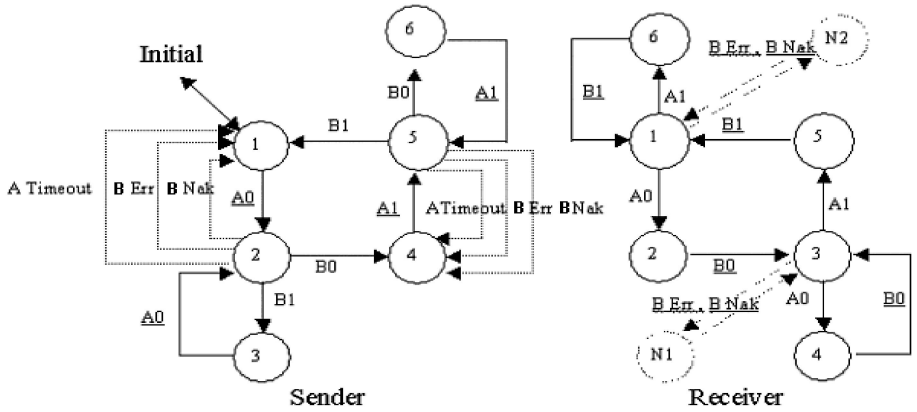
Fig. 6. Discarding packets to handle network congestion. (a) Network congestion occurs, color tokens are injected. (b) Some packets are dropped to reduce traffic. (c) Data transmission is continued after congestion is resolved.

1. Receiver's responses in place p_1: As shown in Fig. 8b, if the receiver (in place p_1) receives message A_0, that means a message is received, a color token used to handle such a situation will be injected into p_1 and transition t_1 will be enabled. Then, B_0 is sent to the sender and t_1 fires—while the resource token moves from p_1 to p_2, the next stage begins. Otherwise, if a Nak or Err signal is detected/received, then the receiver is alerted that there is some problem with communication. A color token used to handle this condition will be injected into p_1, the commands associated with this color token will be executed, a transition, a place, and four arcs will be created represented by the dot lines, as shown in Fig. 8b. Then, transition t_3 fires, the resource token moves to p_3, and Nak/Err is sent to the sender. At last, the resource moves back to p_1—waiting for the message to be resent from the sender.
2. Sender's responses in place p_1: As shown in Fig. 8a, after the sender (place p_1) sent a message A_0 to the receiver, if a timeout, Err, Nak, or B_1 signal is detected/received, then the sender is alerted that transmission has not been successful. Then, a color token used to handle this condition will be injected into p_1, the commands associated with this color token will be executed, a transition t_3 and two arcs, p_1t_3 and t_3p_1, are created represented by the dotted lines shown in Fig. 8a. At last, transition t_3 fires, a resource token is reinjected into place p_1, the current message A_0 will be sent again. If B_0 is received, a color token corresponding to this will be injected into p_1 and transition t_1 will be enabled. At last, transition t_1 fires—while the resource token moves from p_1 to p_2, the next message A_1 is sent.

5 SMCPN Simulator

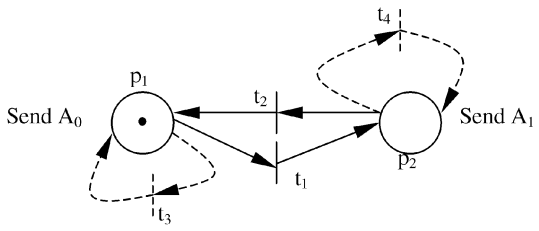
The simulator implemented has two major functions: a presentation specification module and a simulation module. The presentation specification module includes icons of places, arcs, and transitions for programmers to specify temporal relationships among media in presentation. This part is similar to a general Petri net design tool. A more important function is that SMCPN provides the design functions that let users design their own operations. The simulation module offers common user interrupts such as *reverse, skip, and iteration.*

The SMCPN simulator is designed using visual C++ with a user-friendly interface. A user can do most of the job by using the mouse. To draw a place or transition, a user just clicks on the place or transition icons shown on the menu,



Legends:
Solid line: original ABP
Dotted line: self-modifying part

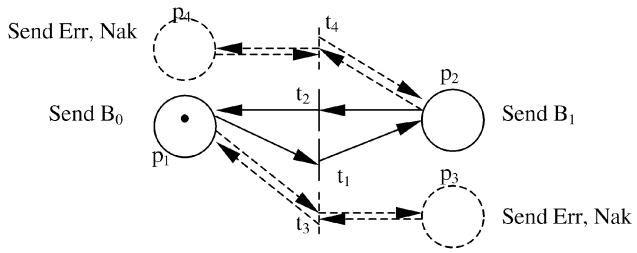
Fig. 7. Extension of the Alternating Bit Protocol.



(a)

Legends:

place	Events (color token injected)	Actions (semantics)
p_1 upon receiving messages/events	Timeout, Err, Nak, B_1	Create transition t_3; Create arcs p_1t_3 & t_3p_1; Enable transition t_3 (Retransmission)
	B_0	Enable transition t_1; (Move token to p_2)
p_2 upon receiving messages/events	Timeout, Err, Nak, B_0	Create transition t_4; Create arcs p_2t_4 & t_4p_2; Enable transition t_4 (Retransmission)
	B_1	Enable transition t_2 (Move token to p_1)



(b)

All modifications will be deleted when the color token finishes its operation & self-deletes. Transitions will be disabled again.

Legends:

place	Events (color token injected)	Actions (Semantics)
p_1 upon receiving messages/events	A_0	Enable transition t_1; (Send Ack, move to p_2)
	Message loss (Nak) or message error (Err)	Create transition t_3; Create Place p_3 Create arcs p_1t_1 & t_1p_3 Create arcs p_3t_3 & t_3p_1 Enable transition t_3; (Send Err or Nak)
p_2 upon receiving messages/events	A_1	Enable transition t_2; (Send Ack, move to p_1)
	Message loss (Nak) or message error (Err)	Create transition t_4; Create Place p_4 Create arcs p_2t_4 & t_4p_4 Create arcs p_4t_4 & t_4p_2 Enable transition t_4; (Send Err or Nak)

Fig. 8. Using SMCPN to design self-modifying ABP. (a) Sender: all transitions are disabled in the beginning. (b) Receiver: all transitions are disabled in the beginning.

as displayed in Fig. 9. A dialogue box will pop up, prompting the user to enter a label. Then, by clicking onto any area within the white screen, a place or transition will be drawn. Also, the places and transitions can be connected together with arcs by clicking the corresponding icon also shown on the menu. After being given an initial marking, the simulator is ready to run. When "Run" is clicked, the simulator will run and fire till no transition is possible.

To simulate user interrupts/network events, we have designed some icons corresponding to different operations (as shown on the menu in Fig. 9). We use a reverse operation as an example to introduce how user interrupt is simulated on our SMCPN simulator. Fig. 9a shows that the last section of a multimedia application model is being presented (the model designed by the user using the SMCPN simulator). At this time, the user wants to reverse it. The only thing he needs to do is to click the "Re" command that simulates the reverse user interrupt. When clicked, a red token corresponding to reverse interrupt will be injected into the place p_5, as shown in Fig. 9b. At the same time, the implementation of the resource token in p_5 is paused.The command associated with the red token will be

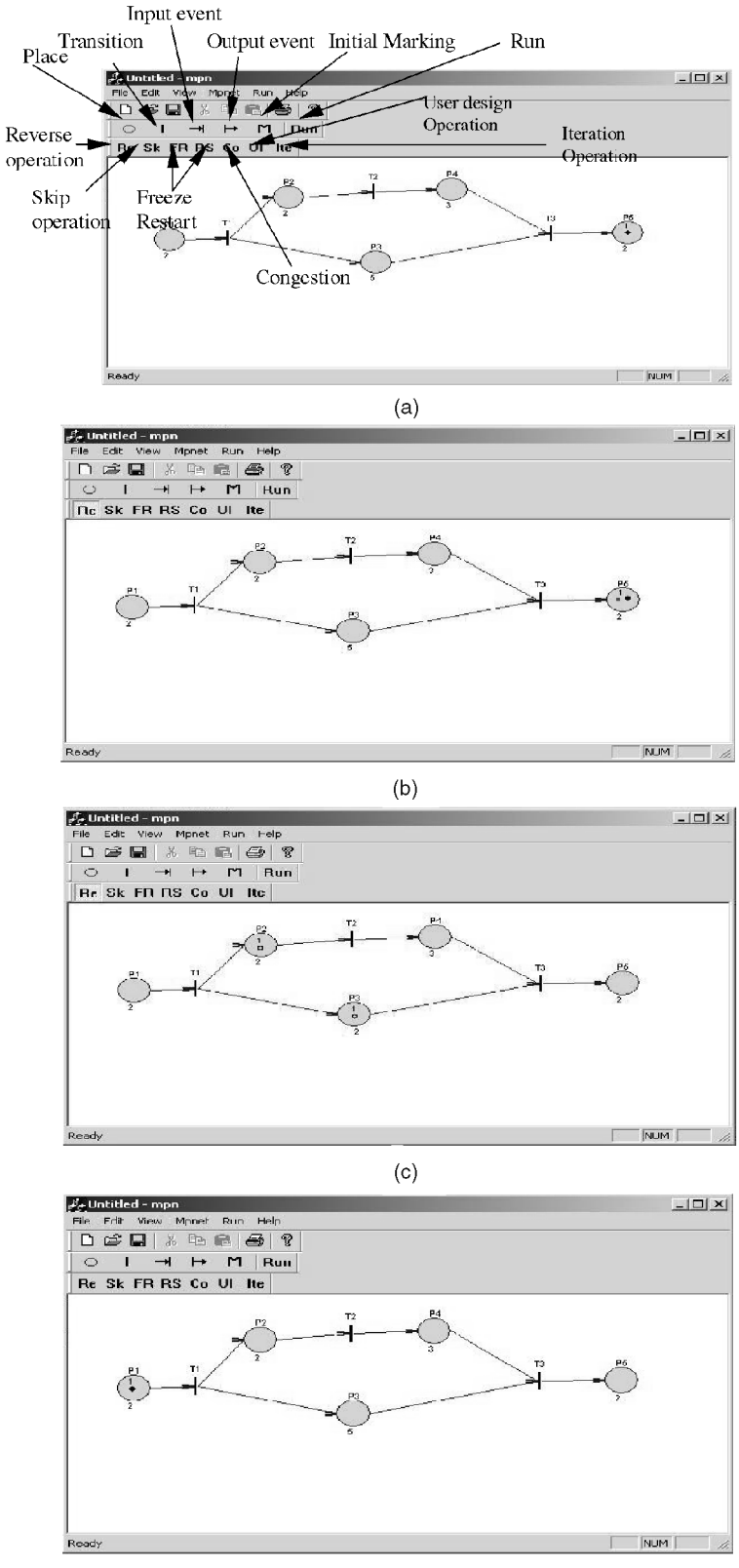


Fig. 9. SMCPN simulator. (a) A token has arrived at the last section. (b) The simulator responds to a reverse interrupt. (c) "Reverse" operation is being executed. (d) The reverse user interrupt finished.

executed, which changes the resource token in p_5 from a forward token to a reverse token. Then, the red token will be deleted. Fig. 9c shows the implementation of this reverse operation is being executed, the resource token moves to the opposite direction of arcs. At last, when the reverse token arrives at p_1 and the implementation finishes, the forward

Fig. 10. Dialogue box—user operation design.

resource token will be restored, as shown in Fig. 9d, the simulation of reverse user interrupt finishes.

We have simulated several user interrupts/network events (e.g., reverse, skip, freeze, resume, and network congestion) in our current prototype and the implementation results confirm the feasibility and general programmability of SMCPN.

Besides these predesigned operations, with the *UI* icon the SMCNP simulator allows users to design their own operations according to their special requirements. Network congestion is used here as an example to show how a user could design his own operation. When a user clicks on the *UI* icon, a dialogue box will pop up, as shown in Fig. 10. The user can input the name of his operation in the *user Operation/Network Event* Name edit box. The number of basic commands to be executed in this operation will be input into the *Number of Commands* edit box. How many times this operation needs to be executed can be specified in the *ExecutionTimes* edit box. Click the *ok* icon after all these parameters have been input. Another dialogue box for the user to specify the sequence of commands will pop up, as shown in Fig. 11. With the number of commands specified as 1 just now, this dialogue box will appear once to let the user input the command label (the commands set as shown in Table 1) according to the execution sequence (if the number is greater than one) that the user has designed. In this particular example, the user specifies a congestion handler by "change duration"—which means "adjust transmission speed" to relieve congestion. Each time the user inputs a command label and clicks the ok button, the dialogue box will display the command sequence, as shown in Fig. 11. When all the command labels have been input and the *ok* icon is clicked, the specification is completed.

6 Conclusions

In this paper, we have proposed Self-Modifiable Color Petri Nets—SMCPN as a powerful tool in handling user interrupts and network events in distributed multimedia systems. By introducing color tokens associated with commands, SMCPN can modify the mechanisms of Petri nets to accomplish real-time user/network operations. SMCPN has the desired general controllability and programmability in the following sense: 1) It allows handling of user manipulations or prespecified events at any time while

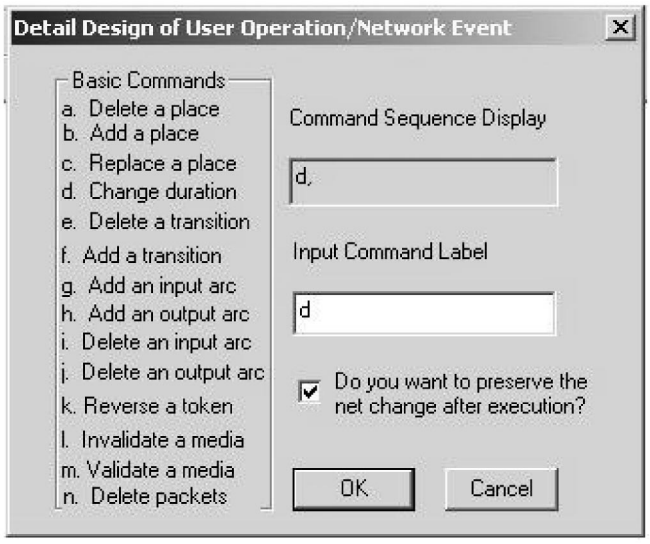


Fig. 11. Dialogue box for a user to customize his own operations.

keeping the Petri net design simple and easy. For existing models to realize the same functions, the design outcome could be messy and the effort is tremendous. 2) It allows the user to customize event handling beforehand. This means the system being modeled can handle not only commonly seen user interrupts (e.g., skip, reverse, freeze), the user is also free to define new operations, including network event handling. 3) It has the power to simulate self-modifying protocols.

SMCPN facilitates a compact and flexible specification of real-time, large-scale synchronization while preserving granularity. By trading verifiability with expressiveness, SMCPN has the desired general programmability when handling user or network interrupts. There is no need for the designers to prespecify individual Petri nets for each place that may be subject to such interrupts. A simulator has been built to demonstrate the feasibility of SMCPN.

Appendix

Skip Operation

Sometimes a user might feel that a certain section of a presentation is boring and want to skip to other sections. The user can choose to skip an on-going stage or skip to a stage that is specified by the user. Assume a multimedia presentation is displaying MTV when a user wants to skip from the current stage to another (p_5 in Fig. 12a) as shown in Fig. 12a. As shown in Fig. 12b, when such a user interrupt occurs, the color tokens corresponding to "skip" will be created in the places which contain resource tokens at the same time that the resource tokens in these places are paused. First, command 1 associated with color token c is executed, transition t_2 will be disabled. Second, command 2 will create transition t_5. Third, command 3 creates the arcs from p_2/p_3 to transition t_5. At last, command 4 creates an arc from transition t_5 to place p_5. Now, the modification of the net is done. The resource tokens in p_2 and p_3 will be resumed and moved through transition t_5 to place p_5, as shown in Fig. 12c. At last, the added parts will be deleted,

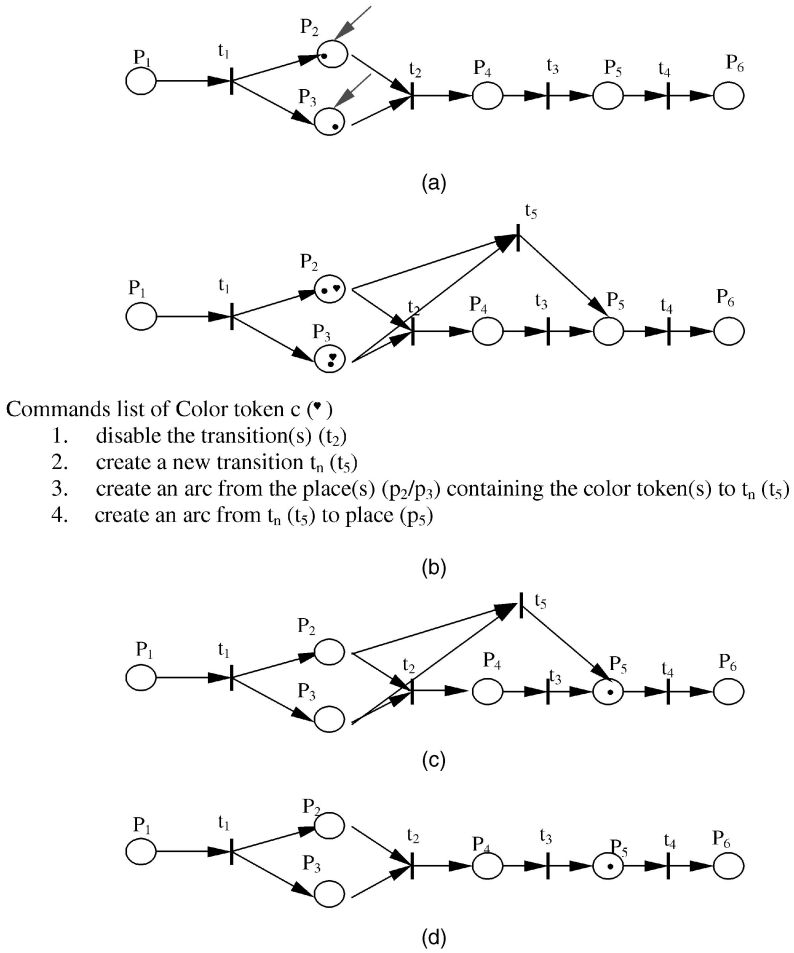


Fig. 12. A skip operation. (a) Skip operation from the current place to p_5. (b) The modification of the model. (c) Skip interrupt in execution. (d) The net structure restored.

the net structure will be restored to the initial state, as shown in Fig. 12d.

Please note that the commands specified in Fig. 12 are generic so that the same specifications apply regardless of where in the net an interrupt might arise. This general programmability is one of the strengths of SMCPN. The same applies to the handling of other user interrupts. Our implementation has confirmed the feasibility of this.

Also the skip operation, when applied to an iterated presentation, can be handled by a color token tagged with a number N_{skip}, as shown in Fig. 13.

The invocation of a skip user interrupt creates the color token associated with the skip operation into the place p_{UI}. It sets the value of N_{skip} to the media segment number to skip to, specified by the user. When the color token for skip is injected into p_{UI}, the current resource segment index will be replaced by N_{skip}. Then, the system will continue its operation from the segment N_{skip}.

REFERENCES

[1] B. Prabhakaran and S.V. Raghavan, "Synchronization Models for Multimedia Presentation with User Participation," *ACM Multimedia Proc.*, pp. 157-166, Aug. 1993.

[2] C.-M. Huang and C.-M. Lo, "An EFSM-Based Multimedia Synchronization Model and the Authoring System," *IEEE J. Selected Areas in Comm.*, no. 1, pp. 138-152, Jan. 1996.

[3] C.-M. Huang and C.-M. Lo, "Synchronization for Interactive Multimedia Presentations," *IEEE Multimedia*, vol. 5, no. 4, pp. 44-62, Oct.-Dec. 1998.

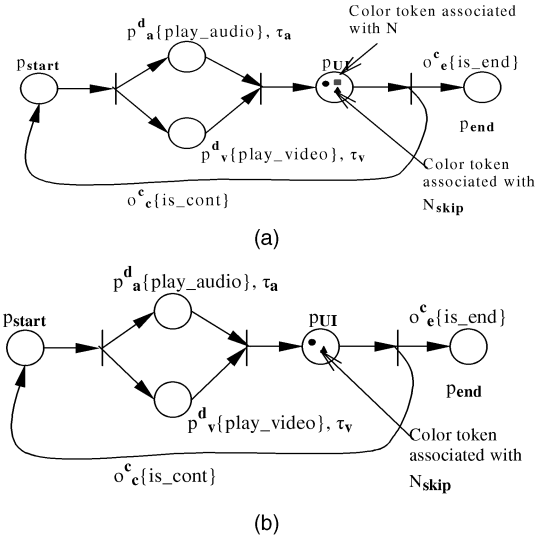


Fig. 13. Multimedia presentation with a skip user interrupt. (a) A color token associated with N_{skip} is injected into p_{UI}. (b) The index color token is replaced with a color token N_{skip}.

[4] C. Nicolaou, "An Architecture for Real-Time Multimedia Communication System," *IEEE J. Selected Areas in Comm.,* vol. 8, no. 3, pp. 391-400, Apr. 1990.
[5] F. Bastian and P. Lenders, "Media Synchronization on Distributed Multimedia Systems," *Proc. Int'l Conf. Multimedia Computing and Systems,* pp. 526-531, 1994.
[6] G. Blakowski and R. Steinmetz, "A Media Synchronization Survey: Reference Model, Specification, and Case Studies," *IEEE J. Selected Areas in Comm.,* vol. 14, no. 1, pp. 5-35, Jan. 1996.
[7] J.L. Peterson, *Petri Net Theory and The Modeling of Systems.* Prentice Hall, 1981.
[8] M. Diaz and P. Senac, "Time Stream Petri Nets A Model for Multimedia Streams Synchronization," *Proc. First Int'l Conf. Multimedia Modeling,* pp. 257-273, 1993.
[9] N.U. Qazi, M. Woo, and A. Ghafoor, "A Synchronization and Communication Model for Distributed Multimedia Objects," *Proc. First ACM Int'l Conf. Multimedia,* pp. 147-155, Aug. 1993.
[10] P.K. Andleigh and T. Kiran, *Multimedia Systems Design,* pp. 421-444. Prentice Hall, 1996.
[11] S.-U. Guan, H.-Y. Yu, and J.-S. Yang, "A Prioritized Petri Net Model and Its Application in Distributed Multimedia System," *IEEE Trans. Computers,* vol. 47, no. 4, pp. 477-481, Apr. 1998.
[12] S.-U. Guan and S.-S. Lim, "Modeling Multimedia with Enhanced Prioritized Petri Nets," *Computer Comm.,* to appear.
[13] T. Agerwala, "A Complete Model for Representing the Coordination for Asychronous Processes," Hopkins Computer Research Report Number 32, Computer Science Program, Johns Hopkins Univ. Baltimore, Md., July 1974.
[14] T.D.C. Little and A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects," *IEEE J. Selected Areas in Comm.,* vol. 8, no. 3, pp. 413-427, Apr. 1990.
[15] W. Reisig, *A Primer in Petri Net Design.* Springer-Verlag, 1992.
[16] K. Jensen, *Coloured Petri Nets,* vol. 1. Springer-Verlag, 1997.
[17] S.-U. Guan and Z. Jiang, "A New Approach to Implement Self-Modifying Protocols," *Proc. 2000 IEEE Int'l Symp. Intelligent Signal Processing and Comm. Systems (ISPACS 2000),* pp. 539-544, Nov. 2000.
[18] Y.Y. Al-Salqan and C.K. Chang, "Temporal Relations and Synchronization Agents," *IEEE Multimedia,* vol. 3, pp. 30-39, 1996.
[19] R. Valk, "On the Computational Power of Extended Petri Nets," *Proc. Math. Foundations of Computer Science 1978,* pp. 526-535, 1978.
[20] K. Rothermel and T. Helbig, "An Adaptive Protocol for Synchronizing Media Streams," *Multimedia Systems,* vol. 5, no. 5, pp. 324-336, 1997.
[21] N.T. Bhatti, M.A. Hiltunen, R.D. Schlichting, and W. Chiu, "Coyote: A System for Constructing Fine-Grain Configurable Communication Services," *ACM Trans. Computer Systems,* vol. 16, no. 4, pp. 321-366, Nov. 1998.
[22] W.T. Tsai, C.V. Ramamoorthy, W.K. Tsai, K. Wei, and O. Nishiguchi, "Adaptive Hierarchical Routing Protocol," *IEEE Trans. Computers,* vol. 38, no. 8, pp. 1059-1075, Aug. 1989.
[23] D.C.A. Bulterman, "SMIL 2.0.2. Examples and Comparisons," *IEEE Multimedia,* vol. 9, no. 1, pp. 74-84, Jan.-Mar. 2002.

Sheng-Uei Guan received the MSc and PhD degrees from the University of North Carolina at Chapel Hill. He is currently with the Electrical and Computer Engineering Department at the National University of Singapore. He has also worked in a prestigious R&D organization for several years, serving as a design engineer, project leader, and manager. He also served as a member on the Republic of China Information & Communication National Standard Draft Committee. After leaving industry, he joined Yuan-Ze University in Taiwan for three and a half years. He served as deputy director for the Computing Center and also as the chairman for the Department of Information and Communication Technology. Later, he joined La Trobe University with the Department of Computer Science and Computer Engineering, where he helped to create a new multimedia systems stream.

Wei Liu received the BEng degree in electrical engineering from Xian Jiaotong University in 1994, the MEng degree in electrical engineering from Xian Jiaotong University in 1997, and the MEng degree in electrical and computer engineering from the National University of Singapore in 2002. Currently, she is a research engineer in the Department of Electrical and Computer Engineering at the National University of Singapore. Her research interests include multimedia systems, augmented and virtual reality, and human computer interaction.