Journal of Computer Science 2 (2): 127-143, 2006
ISSN 1549-3636

The Definition of Extended High-level Timed Petri Nets

Cheng Guangming, Liao Minghong and Wu Xianghu
School of Computer Science and Technology, Harbin Institute of Technology Harbin
People's Republic of China 150001

Abstract: Many extensions of Petri nets have been proposed to model the behaviors and time relations of embedded system, yet these models are all based on some assumptions about the behaviors of embedded systems. Especially they all do not have the actual ability to model interrupt mechanism of embedded system. A new net which is called Extended High Level Timed Petri Nets (EHLTPN) is introduced in this study. It shows how to extend High Level Petri Nets (HLPN) with time, actions and interrupt mechanism. Interruptible subnets corresponding to different interruptible resources are introduced to model the behaviors and time relations of distributed embedded real-time systems. Each interruptible subnet realizes interrupt mechanism by an Interrupt Switch Transition and a set of Resuming Transitions. We give an informal description of this new model and show how this model be formally defined. A transform rule presented shows that each subnet corresponding to an interruptible resource in EHLTPN can be transformed into a behaviorally equivalent subnet of $HLTPN_{AT}$ with priority. This model makes it possible to create the compact and comprehensive models for distributed embedded real-time systems.

Key words: EHLTPN, Petri nets, interrupt, real-time, distributed, embedded system

INTRODUCTION

With the development of electrical technology, embedded system has been widely used in many domains. Their applications range from simple domestic devices, such as washing machines, central heating systems and electronic gatekeepers, to highly critical system, such as flight control, air traffic control, robot control and aero spacecraft etc. There many of them are real-time distributed embedded systems which are related to the health or safety of human life and property. So the safety of such system must be considered.

Time is an important factor which affects the safety of real-time distributed embedded systems. In real-time systems, correctness depends not only on the result produced by computations, but also on the time at which results are produced. The system may enter an incorrect state if the right result is produced too early or too late with respect to certain time bounds[1]. In such case, however, time issues become essential. In the process of designing a real-time embedded system, programmers have to deal with the relating time constrains with the requirement of the system. Especially in distributed real-time embedded systems time relations become more complex than normal embedded system. If the system can't satisfy these time constrains the safety of these systems will be seriously affected.

During the last 20 years, many formal and informal methods have been used for specification, analysis, verification and program testing. Formal methods include standardized specification languages such as the languages ESTELLE[2], LOTOS[3] and SDL[4] and also other widely used specification method, for example, finite automata, Petri net[5] and temporal logics and algebras of processes[6]. They all have the ability to describe time constrains of object system, but most of them focus on the analysis of system specification, many aspects of embedded systems have been omitted, such as interrupt, DMA, operating system. So these methods can't be used to give an actual result of the correctness of system's time relations.

Except Petri net, most of these methods only support static time analysis and can't be used to reflect the active property of embedded system. Especially in distributed real-time embedded system, many processors exchange message currently and the different parts of such system continuously react to the outside environment. These behaviors of such system are active. So Petri net is one of the best methods used to model and analyze distributed real-time embedded system.

Since Carl Adam Petri introduced Petri net in 1962, in order to model different properties of different system, many extensions have been added. Such as, by

Corresponding Author: Cheng Guangming, # 318 School of Computer Science and Technology, Harbin Institute of Technology, Harbin, People's Republic of China 150001, Tel: 086-0451-86413213, Fax: 086-0451-86412241

assigning firing times to the transitions of Petri net time was firstly introduced by Ramchandani in Timed Petri net[7]; In Merlin's time Petri net, an interval is annotated with each transition and transitions only can fire between corresponding intervals after the enable instant of such transition[8]. Time Petri net have more power than timed Petri net. High-Level timed Petri nets (HLTPN) are well described by Ghezzi[9]. HLTPN has the similar time representing mechanism as time Petri net, but in HLTPN, each place can have more than one data types and transitions are annotated with an action inscription to describe the actions of corresponding transition.

At first, Petri net was mainly used to model concurrent system, but now with the development of embedded system, Petri net are more and more used to model embedded system. Some extensions have been introduced, such as Dual Transition Petri Net (DTPN)[10,11]. DTPN captures both control and data flow structure from a behavioral description of an embedded system. It is used to represent the specification of embedded system.

Some efforts also have been done to model the behaviors of embedded operating system and interrupt system. Miguel Felder extends real-time structured analysis to specification of the detailed design of embedded real-time system and combines the proposed notation with Petri nets. In this model, he gives a subnet to model the schedule mechanism of embedded operating system[12]. Giacomo Bucci introduced Preemptive Time Petri Nets by extending Time Petri Nets with an additional mechanism of resource assignment which makes the progress of timed transitions be dependent on the availability of a set of preemptable resources[13,14]. This mechanism corresponds to the preemptable schedule mechanism of most embedded system. Zuberek[15] in Modified M-Timed Petri nets firstly introduces interrupt arcs which can easily be represented with inhibitor arcs[16]. In Modified M-Timed Petri nets, a firing of a transition may be interrupted if the set of this transition's interrupting places are nonempty. If during a firing period of such transition *t*, all *t*'s interrupting places contain at least one token, the firing of *t* ceases and the tokens removed from *t*'s input places at the beginning of firing, are "return" to their original places. Janusz Borkowski gives a descriptive method to model interrupt with Region-based Petri nets (RPN) [17]. As an extension to Color Petri Nets, RPN is proposed to model actions affecting a set of places while the exact marking is neither known nor important.

All of those Petri net extensions mentioned above don't have the ability to model embedded system accurately. Especially they have many difficulties to model the accurate time relations of embedded software. In these extensions, they have all kinds of assumptions about the behaviors of embedded system, such as the events, priority, execute sequence and so on. These assumptions ordinarily just suit to one aspects of embedded software, they lose generality and just adapt to special situation.

Interrupt system is an important part of embedded systems. Different from normal system, embedded systems often have many interrupt sources and interrupts occur continually. Interrupts greatly disturb the program executing sequence and delay the program executing time. So the influence of interrupts must be considered in the development of embedded system. Particularly remarkable is that the schedule mechanisms of most embedded operating systems are based on interrupt mechanism. Embedded operating systems reschedule tasks each time when they exit from interrupt procedure so that embedded operating system can schedule tasks preemptively. Based on interrupt mechanism we can model schedule mechanism easily and directly and needn't relay on those assumptions about embedded system. Hence, Petri nets can perfectly model preemptive schedule of tasks as long as it can model interrupt.

Distributed real-time embedded systems are complex real-time systems. Such system normally is composed of many embedded processors and run different embedded software. These processors exchange data each other and communicate with outside environment concurrently through all kinds of interfaces. So the time relations of such system are more complex than single processor embedded system. Until now there hasn't a model that can descript the active time relations of such system perfectly.

In this study, we introduce an extension of HLPN[18], called Extended High-Level Timed Petri Net, which allows the representation of the time relations and resource management of complex distributed real-time embedded system in a simple and direct way. This model extends HLPN by adding time, actions and interruptible subnets. Each interruptible subnet corresponds to an interruptible resource and includes an Interrupt Switch Transition and a set of Resuming Transitions. These two special type transitions are used to model the behaviors of each interruptible subnet when interrupts occur. Formal and informal definitions of EHLTPN are all presented. A rule for transforming an interruptible subnet into $HLTPN_{AT}$ with priority is also presented. At last, give a small example of EHLTPN.

Informal definition of EHLTPN: Traditional Petri Net has many difficulties in modeling interrupt. First of all, interrupts occur irregularly, they perhaps happen at any point allowing the occurrence of interrupt. So we can't know the exact point where interrupt occurs. All of the points that have the possibility to be interrupted should

be considered. Secondly, if model all the points where interrupts perhaps happen, the number of place will be enormous. It's hard to manage this model. Thirdly, the analysis of such model is very difficult.

An example of interrupt model of HLPN with time and priority is shown in Fig. 1. In Petri net with priority, each transition is annotated with a natural number, called priority level. If more than two transitions are enabled at the same time, only those annotated with highest priority level are allowed to fire. The firing time of transitions is denoted by notation "d" and "d" is annotated with each transition. This figure describes an interrupt model of a program section in which interrupts perhaps happen at place Point(i) and Point(i+1). Except transition Point(i)SwProcA, Point(i)SwProcB, Point(i+1)SwProcA and Point(i+1)SwProcB, all the transitions in this model have priority level 1 while the former four transitions have higher priority level 2. The left part of this figure models a program section. Each transition in this section represents one time execution of code and at the moment when these transitions fires, the processes of firing can't be interrupted. This means that interrupt only can occur where the program section run to a place otherwise the interrupt will not occur.

Place INTNo represents the source of interrupts. If there is a token in it, shows that an interrupt request has been pended. Place INTTable keeps a list to save the interrupt procedure entrance of each interrupt number. Place INTProcA and INTProcB represent the beginning of two interrupt procedures. Transition Point(i)SwProcA, Point(i)SwProcB, Point(i+1)SwProcA and Point(i+1)SwProcB represent the behaviors when interrupts occur at Point(i) or Point(i+1). Transition ProcAIRETPoint(i), ProcAIRETPoint(i+1), ProcBIRETPoint(i) and ProcBIRETPoint(i+1) represent the operations when interrupt procedure ends. Place Point(i)RevA, Point(i)RevB, Point(i+1)RevA and Point(i+1)RevB are used to keep the status of Point(i) and Point(i+1) corresponding to different interrupt procedures when interrupt occurs and used to restore the status of Point(i) or Point(i+1) when interrupt procedure ends.

Suppose at a moment, interrupt 1 comes, thus a token emerges in place INTNo and its value is 1. At the same time, program section 1 run to place Point(i), so transition t(i+1) and transition Point(i)SwProcA are enabled at the same time. However, transition Point(i)SwProcA has the higher priority level than transition t(i+1), so it fires and prevents the firing of transition t(i+1). Transition Point(i)SwProcA from place INTTable gets the interrupt procedure entrance of interrupt number 1, place INTProcA. Then executes Procedure A and keep the status of place Point(i) in place Point(i)RevA. When this procedure ends, transition ProcAIRETPoint(i) restore the status of point(i). The process of interrupt finished.

From Fig. 1 we can see that the numbers of places and transitions are directly proportional to the number of interruptible points in the model. An embedded software model perhaps exists millions of such points. Fortunately we needn't know the exact point where an interrupt occurs. Knowing that the interrupt occurs in which program section is enough.

In this study, EHLTPN extended from HLPN is introduced. In EHLTPN, places represent the different states of system; tokens in place represent the resources of system; transitions represent the operations that operate on input tokens. Each token can save arbitrary complex data types of data, normally represented by a tuple, hence one token can represent many kinds of resource at the same time. In EHLTPN, a time function is annotated with each transition indicating how long the input tokens are kept in this transition after the transition fires. An action inscription which models the behavior of each transition is annotated with each transition.

In architecture, EHLTPN contains a finite number of interruptible subnets. Each interruptible subnet corresponds to an interruptible resource. This resource takes part in all the operations of this subnet and it is the unique interruptible resource in this interruptible subnet. At certain condition it can interrupt current operation and then goes to run another series of operations and when these operations end, it resumes the status of the operation interrupted just now and continues to finish this operation just like it hasn't been interrupted. Such procedure is called interrupt procedure.

Interruptible resource has two forms in each subnet. One form is a token whose data type is a tuple with two elements and the content of its first element is a sign which is used to represents the existence of interruptible resource in this subnet. Places of such data type are called interruptible places. Another form is a firing transition. Because interruptible resource takes part in all the operations of this subnet and it is the unique interruptible resource in this subnet, each transition in this subnet has one and only one interruptible place belong to its preset and the cardinality of the multiset annotated with the arc from this interruptible place to the transition is one. Similarly, for each transitions in this subnet, has one and only one interruptible place belong to its postset and the cardinality of the multiset annotated with the arc from this transition to the interruptible place is one too. Hence when a transition fires, it moves a token with interruptible resource from an interruptible place belongs to its preset. When the transition stops firing, it also moves a token with interruptible resource to an interruptible place belongs to its postset.

An active node is a node which has the interruptible resource in an interruptible subnet. It may

be a place or a transition. An interruptible place that has a token is an active node. A firing transition also is an active node. All the active nodes except uninterruptible transitions are interruptible. Since the interruptible resource in a subnet is unique, at any time, in each subnet, exists one and only active node. Thus when an interrupt occurs, we can get the active node which is interrupted directly. The status of this active node at the moment when it is interrupted is called a breakpoint.

Each interruptible subnet has two special type transitions, called Interrupt Switch Transition and Resuming Transition. Interrupt Switch Transition represents the operations when interrupt occurs and Resuming Transition represents the operations when the interrupt procedure ends. Each interruptible subnet has only one Interrupt Switch Transition but may have a group of Resuming Transitions. Thus each interruptible subnet only can responses to one interrupt request a time but may have many interrupt procedures concurrently.

At the moment when Interrupt Switch Transition fires, it saves the breakpoint of the interrupted active node to a special variable *bp* associated with it and the data type of this variable is "BreakPoint". Interrupt Switch Transition interrupts the operations of current active node at the same time. If this breakpoint is a place, then the token in it is removed; if the breakpoint is a transition, set the remaining firing time of this transition 0 and the status of this transition recovers to be "normal" just like it has not fired. At the moment when Interrupt Switch Transition stops firing, it resumes the breakpoint which is produced by the function B annotated with it. When Resuming Transition stops firing, it does the same operation as Interrupt Switch Transition does. The breakpoint may be a place or a transition. If it is a place, then move the token keeping in the breakpoint to it. If it is a transition then resume its status kept in the breakpoint. This status includes the remaining firing time of this transition, the values binding to the variables annotated with its input arcs and tokens which are moved in as it fired. But these tokens in most cases have no use for the firing of this transition, so in the latter of this study we don't keep the values of these tokens.

When we use EHLTPN model distributed real-time embedded system, each subnet corresponds to the software running on an embedded processor. The processor is the interruptible resource and takes part in all the operations of all the transitions. Interrupt Switch Transition corresponds to operations when the processor responds to the interrupt requests from outside and Resuming Transition corresponds to the machine instruction IRET.

Figure 2 is an interrupt model represented by EHLTPN corresponding Fig. 1. In this model, the arcs drawing by dash style are not the true arcs of this model. Here, they are used to represent how the interruptible resource is moved when interrupt occurs.

Formal definition of EHLTPN

The definition of EHLTPN: At first, we recall the definition of HLPN. All the notations without clear definition here refer to[18].

Definition 3.1: HLPN is a structure HLPN=(NG, S, C, AN, M_0) where

(i). NG=(P, T, F) is called net graph with

* P a finite set of nodes, called Places. ID:$P\rightarrow N$, is a function marking P, N=(*1*,*2*,...) is the set of natural number. Using p_1, p_2, ..., p_n represents the elements of P and n is the cardinality of set P;

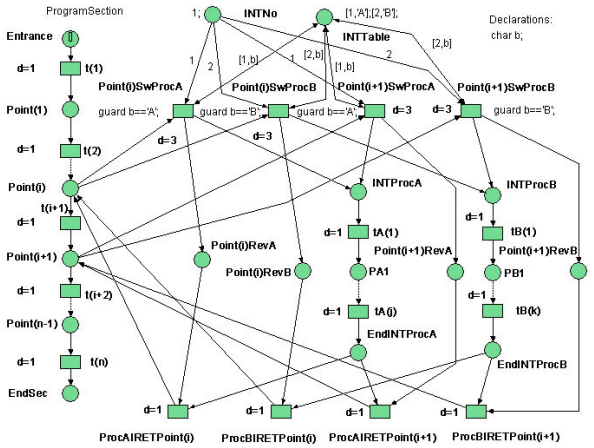

Fig. 1: An interrupt model represented by HLPN with time, actions and priority

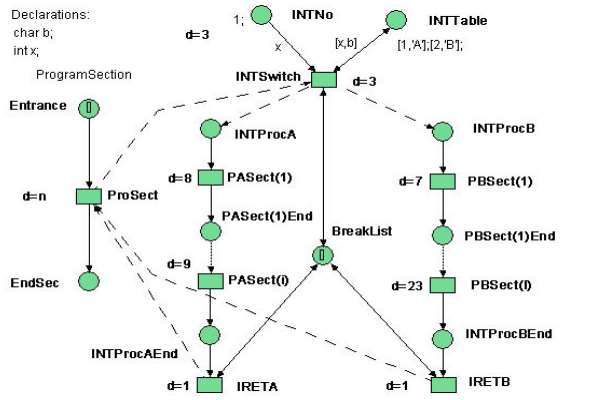

Fig. 2: An interrupt model represented by EHLTPN corresponding to Fig. 1

* T a finite set of nodes, called Transitions, which disjoint from P, $P \cap T = \varnothing$; ID:$T\rightarrow N$ is a function marking T. Using t_1, ..., t_m represents the elements of T, m is the cardinality of set T;

* $F \subseteq (P \times T) \cup (T \times P)$ a set of arcs, known as the flow relation;

(ii). $\sum=(R, \Omega, V)$ is a Natural-Boolean signature with variable. It has a corresponding $\sum$-Algebra, $H=(R_H, \Omega_H)$. Where,

* R is a set of sorts;
* Ω is a set of operators together with their arity in R which specifies the names of the domain and co-domain of each of operators;
* V is a set of sorted variables;

(iii). $C:P \rightarrow R_H$ is a function which types places;

(iv). $AN=(A, TC)$ is a pair of net annotations;

* $A:F \rightarrow BTERM(\Omega \cup V)$ such that for $C(P)=H_r$ and for all (p,t), $(t',p) \in F$, $A(p,t)$, $A(t'p) \in BTERM(\Omega \cup V)_r$. A is a function that annotates arcs with a multiset of terms of the same sort as the carrier associated with arc's place;
* $TC:T \rightarrow TERM(\Omega \cup V)_{Bool}$ is a function that annotates transitions with Boolean expressions;

(v). $M_0:P \rightarrow \bigcup_{p \in P} \mu C\ (p)$ such that $\forall\ p \in P, M_0(p) \in \mu C(p)$ is the initial marking function which associates a multiset of tokens of correct type with each place.

Then we give the definition of HLPN with time and actions.

Definition 3.2: High-level Petri Nets with actions and time, $HLPN_{AT}$, is a structure, $HLPN_{AT}=(NG, S, C, AN, AC, D, M_0^*)$, where

(i). $HLPN=(NG, S, C, AN, M_0)$;

(ii). $D: T \rightarrow TERM(\Omega \cup V)_{R^{\geq 0}}$, D is a function that annotates transitions with a rational number representing the initial remaining firing time of transitions. It is initiated at the moment when the transition begins to fire. For convenience's sake, use set $D=\{d_1, \ldots, d_m\}$ to represent the initial remaining firing times corresponding to $T=\{t_1, \ldots, t_m\}$ respectively. $R^{\geq 0}$ is the set of nonnegative rational numbers;

(iii). $AC:T \rightarrow BTERM(\Omega \cup V)$, AC is a function that annotates transitions with a multiset of terms. $AC(t)$ represents the actions of transition t;

(iv). $M_0^*=(M_0, r_0)$, is a pair, represents the status of net at time 0 (we assume that each net start at time 0). Where,

* $M_0:P \rightarrow \bigcup_{p \in P} \mu C\ (p)$, is the initial marking function which associates a multiset of tokens with each place at time 0;
* $r_0:T \rightarrow 0$, is a function representing the remaining firing time of each transition. The remaining firing times of all the transitions evaluate to 0 at time 0.

The set of interruptible places in a interruptible subnet $N_{C_j} \in NC$ is denoted by P_{IC_j}. The set of active places which have a token in subnet N_{C_j} is denoted by P_{RunC_j}. The capacities of each place belonging to P_{IC_j} is 1, because in N_{C_j} only exists one interruptible resource C_j. T_{Run} is the set of transitions which are firing at current time, $T_{Run}=\{t \mid t \in T \wedge r_{current}(t)>0\}$. If $t \in T_{Run}$ calls transition t being "running" status. T_{RunC_j} is the set of transitions which are firing in subnet N_{C_j}, $T_{RunC_j}=\{t \mid t \in T_{C_j} \wedge r_{current}(t)>0\}$. The set of active nodes in N_{C_j} is denoted by N_{RunC_j}, $N_{RunC_j} = T_{RunC_j} \cup P_{RunC_j}$. Later, in Theorem 3.1 gives that at any time $\forall\ N_{C_j} \in NC, \mid N_{RunC_j} \mid =1$. So at any time when an interrupt occurs, variable bp may have a unique value.

In EHLTPN a special data type "BreakPoint" is introduced. BreakPoint is used to save the status of the breakpoint where interrupt occurs. Breakpoint may be a place or a transition. If the breakpoint is a place, the type of this breakpoint, the ID of this place and the tokens in this place are kept. If it is a transition, the type of this breakpoint, the ID of this transition, the left firing time of this transition and the values of variables annotated with the input arcs of this transition are kept.

Because only variables which have bound values can take part in the calculation of actions and the terms annotated with output arcs, thus keep the values of every variables is enough.

In order to give a clear definition of the semantic of EHLTPN, here give some definitions of notations about data type BreakPoint. Variables of BreakPoint data type usually are denoted by *bp* or bp_1, bp_2, bp_3, …. Because of the complexity of BreakPoint data type, it can't be described by normal data type. In order to get and set the values of a BreakPoint data type, following methods are provided:

* *bp.type*() returns the type of breakpoint *bp*. If returns *t*, represents *bp* is a breakpoint of transition type. Otherwise it will return *p* to represent it is a place type;
* *bp.settype*(*p or t*) sets the type of *bp*;
* *bp*,*ID*()returns the node *ID* of breakpoint *bp*. For example, if the *bp* is a breakpoint of transition t_9, then *bp.ID*()=*9*;
* *bp.setID*(*N*), sets the node *ID* of breakpoint *bp*;
* *bp.token*(), if *bp.type*()=*p*, returns the multiset of place $p_{bp.ID()}$, $M(p_{bp.ID()})$; otherwise returns $\varnothing$;
* *bp.settoken*(*M*(*p*)), saves the multiset of place;
* *bp.rt*(),if *bp.type*()=*t*, returns the remaining firing time of transition $t_{bp.ID()}$;
* *bp.setrt*(*r*(*t*)), sets the remaining firing time of transition *t*;
* *bp.f*(*var*), is a function constructed by *bp.Conf*(var_1, var_2, …, var_n, val_1, val_2, …, val_n), it will return the value of variable *var*;
* *bp.Conf*(var_1, var_2, …, var_n, val_1, val_2, …, val_n), according the input parameter constructs a function *f* to save the value of each variables. $\forall var_m \in (var_1, var_2, \ldots, var_n)$, $1 \le m \le n$, $bp.f(var_m)=val_m$.

Definition 3.3: Extended high-level Timed Petri Nets is a tuple EHLTPN=($HLPN_{AT}$, PC, NC), where

(i). $HLPN_{AT}$ is a high-level Petri nets with actions and time, $HLPN_{AT}$ =(*NG*, *S*, *C*, *AN*, *AC*, *D*, M_0^*);

(ii). $PC=\{C_1, C_2, \ldots. C_k\}$, $K>0$, $k \in N$, is a set of interruptible resources;

(iii). $NC=\{N_{C_1}, N_{C_2}, \cdots N_{C_k}\}$, $C_1, C_2, C_k \in PC$, is the set of interruptible subnets of EHLTPN and $\forall\ N_{C_j} \in NC$, $1 \le j \le k$, $N_{C_j}=(P_{C_j}, T_{C_j}, C_{C_j}, B_{C_j}, t_{SC_j}, T_{RC_j}, bp_{C_j})$ is a structure, where

* P_{C_j} is the set of places in N_{C_j}, $P_{C_j} \subseteq P$, $P_{C_j}=P_{IC_j} \cup P_{NIC_j}$, $P_{IC_j} \cap P_{NIC_j}=\varnothing$; P_{IC_j} is the set of interruptible places in P_{IC_j}; P_{NIC_j} is the set of uninterruptible places in P_{C_j};
* T_{C_j} is the set of transitions in N_{C_j}. $T_{C_j} \subseteq T$ and $T_{C_j}=T_{IC_j} \cup T_{NIC_j}$, $T_{IC_j} \cap T_{NIC_j}=\varnothing$, $\forall\ N_{C_i}, N_{C_j} \in NC$, $N_{C_i} \cap N_{C_j}=\varnothing$; T_{IC_j} is the set of interruptible transitions in T_{C_j}; T_{NIC_j} is the set of uninterruptible transitions in T_{C_j};
* $C_{C_j}: P_{IC_j} \to R_{HC_j}$, is a function which types places in P_{IC_j}. $R_{HC_j}=\{H_{rC_j} \mid r \in R\}$, $H_{rC_j}=\{(C_j, r) \mid r \in H_r\}$;

$$\forall\ p \in P_{C_j},\quad C(p)=\begin{cases} C(p) & p \in P_{NIC_j} \\ C_{C_j}(p) & p \in P_{IC_j} \end{cases}$$

* $B_{C_j}: \{t_{SC_j}\} \cup T_{RC_j} \to TERM(\Omega \cup V)_{BreakPoint}$ is a function annotated t_{SC_j} and elements in T_{RC_j}. This function gives the breakpoints of resuming after the firing of these transitions;
* t_{SC_j} is Interrupt Switch Transition, $t_{SC_j} \in T_{NIC_j}$;

* T_{RC_j} is the set of Resuming Transitions, $T_{RC_j} \subset T_{NIC_j}$;

* bp_{C_j} is a variable, $bp_{C_j} \in V$. At the moment t_{SC_j} fires, bp_{C_j} is binding to the value of the breakpoint of current active node which is interrupted. This variable only can be used in the actions annotated with t_{SC_j}, and the terms annotated with the output arcs of t_{SC_j}.

Definition 3.4: The definition of initial marking, M_0^*, of EHLTPN is the same as $HLPN_{AT}$, however because in each subnet only exist one interruptible resource, it must satisfy following property:

$$\forall\ N_{C_j} \in NC, \sum_{\forall p \in P_{IC_j}} |M_0(p)| = 1.$$

Definition 3.5: The marking, M_{time}^*, of EHLTPN is defined in the same way as the initial marking representing the status of net at time *time*. M_{time}^* $=(M_{time}, r_{time})$, where

(i). $M_{time}: P \rightarrow \bigcup_{p \in P} \mu C(p), \forall p \in P, M_{time}(p) \in \mu C(p)$;

(ii). $r_{time}: T \rightarrow R^{\geq 0}$, r_{time} is a function representing the remaining firing time of each transition.

If transition $t \notin T_{Run}$, $r_{time}(t) = 0$, otherwise $r_{time}(t) \neq 0$ and the value of it will decrease automatically as the time passing. The remaining firing time of all the breakpoints of type "t" keep no change until they are resumed and being an active node again. When the remaining firing time of transition t decreases to 0, this transition stops firing and produces output tokens and become an inactive node.

In EHLTPN the definition of preset and postset is the same as other kind of Petri Nets. That is, $\forall\ x \in P \cup T$,

$$^\bullet x = \{y \mid (y \in P \cup T) \wedge ((y,x) \in F)\},$$

and

$$x^\bullet = \{y \mid (y \in P \cup T) \wedge ((x,y) \in F)\},$$

Call $^\bullet x$ and $x^\bullet$ the preset and postset separately.

Behavior of EHLTPN

Definition 3.6: Enabling rule: a transition $t_i \in T \wedge t_i \notin T_{Run}$ is enabled in Marking, M_{time}^*, at time *time*, for a particular assignment to its variables, α_{time}, known as a mode of t_i, iff following properties are satisfied:

(i). $t_i \in T$, $assign_{bool}(TC(t_i)) = true$, $\forall p \in P$,

$Val_{\alpha_{time}}(\overline{p,t_i}) \leq M_{time}(p)$, where for

$(u,v) \in (P \times T) \bigcup (T \times P)$,

* $(u,v) \in F$, $\overline{u,v} = A(u,v)$,

* $(u,v) \notin F$, $\overline{u,v} = \Phi$ and $Val_{\alpha_{time}}(\Phi) = \phi$, the empty multiset;

(ii). $\forall N_{C_j} \in NC$, $t_i \in T_{C_j} \wedge (t_i \notin (\{t_{SC_j}\} \cup T_{RC_j}))$,

satisfy (i) and $\exists p((p \in P_{IC_j} \cap {}^\bullet t_i) \wedge |Val_{\alpha_{time}}(\overline{p,t_i})|$

$= 1) \wedge |P_{IC_j} \cap {}^\bullet t_i| = 1$;

(iii). $\forall N_{C_j} \in NC$, $t_i = t_{SC_j}$, $t_{SC_j} \in T_{C_j}$, satisfy (i) and

$(\exists p(p \in P_{IC_j} \wedge |M_{time}(p)| = 1) \vee \exists t(t \in T_{RunC_j} \wedge t \notin T_{NIC_j}))$

$\wedge \neg \exists p'((p' \in P_{IC_j} \cap {}^\bullet t_{SC_j}) \wedge |Val_{\alpha_{time}}(\overline{p',t_{SC_j}})| \geq 1)$;

(iv). $\forall N_{C_j} \in NC$, $t_i \in T_{RC_j}$, satisfy (ii).

To give the abstract definition about time, here a well-defined semantics is given:

t^+ and t^-, are two special time relative to time t, $(t^+ > t) \wedge \neg \exists t'(t < t' < t^+)$ and $(t^- < t) \wedge \neg \exists t'(t^- < t' < t)$.

Definition 3.7: Priority rule: In each subnet $\forall N_{C_j} \in NC$, when t_{SC_j} and other transitions $t \in T_{C_j}$ are enabled at the same time, t_{SC_j} has the priority of firing.

Following definition 3.6, transition t_{SC_j} and other transitions $t \in T_{C_j}$ are able to be enabled at the same time. But in each interruptible subnet $\forall N_{C_j} \in NC$ only exists one interruptible resource, if $t \in T_{C_j}$ fires, it would prevent the firing of t_{SC_j}. Because the function of transition t_{SC_j} is to interrupt the current running procedure of this subnet, so t_{SC_j} should fire firstly. Hence t_{SC_j} must have higher priority of firing than any other transitions in T_{C_j}.

Definition 3.8: Transition rule:

(i). At time *time*, if transition $t_i \in T$ is enabled in mode α_{time}, for marking M^*_{time}, t_i may occurs in mode α_{time}, at this moment, the value of d_i is evaluated:

$d_i = assign_{R^{\geq 0}}(D(t_i))$

At the beginning of firing, a tuple of tokens which enable t_i is moved from ${}^\bullet t_i$ to t_i, the marking of the net is transformed to a new marking $M^*_{time^+}$ at time $time^+$, denoted by $M^*_{time}[t_i, \alpha_{time}\rangle M^*_{time^+}$, according to the following rule:

For $\forall p \in P$,

$$M_{time^+}(p) = M_{time}(p) - Val_{\alpha_{time}}(\overline{p,t_i}) - \sum_{t \in ((Ts_{time} - \{t_i\}) \cap p^\bullet)} Val_{\alpha_{time}}(\overline{p,t}), \text{ and}$$

$$r_{time^+}(t_i) = d_i - (time^+ - time) \approx d_i$$

$Ts_{time} = \{t \mid t \in T \wedge r_{time}(t) = 0 \wedge r_{time^+}(t) > 0\}$ is the set of transitions in T which start firing at time *time*. $Te_{time} = \{t \mid t \in T \wedge r_{time}(t) = 0 \wedge r_{time^-}(t) > 0\}$ is the set of transitions which stop firing at time *time*.

These inputted tokens are kept in transition t_i for time d_i, until the remaining firing time of transition t_i decreases to 0. Then calculates the terms in actions which annotated with transition t_i and produces output tokens according to the inscriptions annotated with its output arcs, the marking of the net is transformed to a new marking $M^*_{time+d_i}$ denoted by $M^*_{time}[t_i, \alpha_{time}\rangle M^*_{time+d_i}$, according to the following rule:

For $\forall p \in P$,

$$M_{time+d_i}(p) = M_{(time+d_i)^-}(p) + Val_{\alpha_{time}}(\overline{t_i, p}) + \sum_{t_k \in ((Te_{(time+d_i)} - \{t_i\}) \cap {}^\bullet p)} Val_{\alpha_{time_k}}(\overline{t_k, p}) \text{ and}$$

$r_{(time+d_i)}(t_i) = 0$,

where, α_{time_k} is the model which enable transition t_k at time $time_k$ and t_k occurs in this model.

(ii). At time *time*, $\forall N_{C_j} \in NC$, $t_i \in T_{C_j} \wedge (t_i \notin (\{t_{SC_j}\} \cup T_{RC_j}))$, if transition $t_i \in T_{NIC_j}$ is enabled in mode α_{time}, for marking M^*_{time}, then t_i may occurs in mode α_{time} according rule (i) and following property should be satisfied:

$$\exists p (p \in (P_{IC_j} \cap t_i^\bullet) \wedge | Val_{\alpha_{time}}(\overline{t_i, p}) | = 1) \wedge | P_{IC_j} \cap t_i^\bullet | = 1$$

; If transition $t_i \in T_{IC_j}$ is enabled in mode α_{time}, for marking M^*_{time}, then at time $time$, t_i may occurs in mode α_{time} according rule (i). Suppose the remaining firing time of transition t_i decrease to 0 at time $time'$. Then calculates the terms in actions which annotated with transition t_i and produces output tokens according to the inscriptions annotated with its output arcs, the marking of the net is transformed to a new marking $M^*_{time'}$ denoted by $M^*_{time}[t_i, \alpha_{time}\rangle M^*_{time'}$, according to the following rule:

For $\forall p \in P$

$$M_{time'}(p) = M_{(time')^-}(p) + Val_{\alpha_{time}}(\overline{t_i, p}) + \sum_{t_k \in ((Te_{time'} - \{t_i\}) \cap {}^\bullet p)} Val_{\alpha_{time_k}}(\overline{t_k, p}) \text{ and}$$

$r_{(time')}(t_i) = 0$,

Because transition t_i may be interrupted and be resumed in the procedure of firing , so $time' \geq time + d_i$. Following property also should be satisfied:

$$\exists p(p \in (P_{IC_j} \cap t_i^\bullet) \wedge |Val_{\alpha_{time}}(\overline{t_i, p})| = 1) \wedge |P_{IC_j} \cap t_i^\bullet| = 1$$

;

(iii). At time $time$, $\forall N_{C_j} \in NC$, $t_i = t_{SC_j}$, if transition $t_{SC_j} \in T_{C_j}$ is enabled, for marking M^*_{time}, t_{SC_j} may occurs in mode α_{time} according rule (i) and does following operations:

a). At time $time^+$, evaluates bp_{C_j} the value of breakpoint to keep the status of breakpoint;
If breakpoint is P_m, then,

$bp_{C_j}.settype(p)$; $bp_{C_j}.setID(m)$;

$bp_{C_j}.settoken(M_{time}(p_m))$;

If breakpoint is t_m, then

$bp_{C_j}.settype(t)$; $bp_{C_j}.setID(m)$;

$bp_{C_j}.setrt(r_{time}(t_m))$; $bp_{C_j}.Conf(var_1, var_2, \cdots, var_n, \alpha_{time_m}(var_1), \alpha_{time_m}(var_2), \cdots, \alpha_{time_m}(var_n))$,

$\{var_1, var_2, \cdots, var_n\} = \bigcup_{p \in {}^\bullet t_m} Var(A(p, t_m))$;

$Var(terms)$ is the set of variables in $terms$.

α_{time_m} is the mode which enable transition t_m.

b). At time $time^+$, interrupts the operations of current active node. If current active node is p_m, then moves all the tokens in p_m, $M_{time^+}(p_m) = \Phi$; If current active node is t_m, then $r_{time^+}(t_m) = 0$, recovering its status to "normal" status just like it has not fired.

c). At time $time + d_{SC_j}$, resumes the breakpoint $Bp_{C_j}(t_{SC_j})$.

If $Bp_{C_j}(t_{SC_j}).type() = p$ then

$$M_{(time+d_{SC_j})}(p_{BP_{C_j}(t_{SC_j}).ID()}) = M_{(time+d_{SC_j})^-}(p_{BP_{C_j}(t_{SC_j}).ID()}) + Bp_{C_j}(t_{SC_j}).token()$$

;

If $Bp_{C_j}(t_{SC_j}).type() = t$ then

$r_{(time+d_{SC_j})}(t_{Bp_{C_j}(t_{SC_j}).ID()}) = Bp_{C_j}(t_{SC_j}).rt()$ and,

$\forall var \in \bigcup_{p \in {}^\bullet t_{Bp_{C_j}(t_{SC_j}).ID()}} Var(A(p, t_{Bp_{C_j}(t_{SC_j}).ID()}))$,

$var = Bp_{C_j}(t_{SC_j}).f(var)$.

(iv). At time $time$, $\forall N_{C_j} \in NC$, if $t_i \in T_{RC_j}$ is enabled in mode α_{time}, for marking M^*_{time}, t_i may occurs

in mode α_{time}, according rule (i) and at time $time + d_i$ resumes breakpoint $Bp_{C_j}(t_i)$:

If $Bp_{C_j}(t_i).type() = p$ then

$$M_{(time+d_i)}(P_{Bp_{C_j}(t_i).ID()}) = M_{(time+d_i)^-}(P_{Bp_{C_j}(t_i).ID()}) + Bp_{C_j}(t_i).token();$$

If $Bp_{C_j}(t_i).type() = t$ then

$$r_{(time+d_i)}(t_{Bp_{C_j}(t_i).ID()}) = Bp_{C_j}(t_i).rt()$$ and

$$\forall var \in \bigcup_{p \in {}^\bullet t_{Bp_{C_j}(t_i).ID()}} Var(A(p, t_{Bp_{C_j}(t_i).ID()}))$$

$$var = Bp_{c_j}(t_i).f(var).$$

Theorem 3.1: $\forall N_{C_j} \in NC$, at any time, $|N_{RunC_j}| = 1$.

Proof: The proof is by induction on time. For the inductive step, we need to consider two cases.

Case 1: At time 0, by definition 3.4 $\forall t \in T_{C_j}$, $r_0(t) = 0$ and $\forall N_{C_j} \in NC, \sum_{\forall p \in P_{IC_j}} |M_0(p)| = 1$. So all the transitions are inactive and there is only one place $p \in P_{IC_j}$ which has a token with interruptible resource. Since $T_{RunC_j} = \varnothing$, $|P_{RunC_j}| = 1$ therefore, $|N_{RunC_j}| = 1$ obviously holds.

Case 2: At time $time > 0$, if $\forall t \in T_{C_j}$, none of these transitions fire, $M_{time} = M_0$, from case 1, $|N_{RunC_j}| = 1$ is immediate. By Definition 3.6, at any time, $\forall t \in T_{C_j}$ except t_{SC_j} exist and only exist one $p \in P_{IC_j} \cap {}^\bullet t$ and $|Val_{\alpha_{time}}(\overline{p,t})| = 1$, thus when t fires, it will move a token with interruptible resource from p. Because at time 0 $|P_{RunC_j}| = 1$, hence only one transition can fire, thus at this moment $|T_{RunC_j}| = 1$. At the moment, time $time'$, transition t stops firing, according Definition 3.8, exist and only exist one place $p \in P_{IC_j} \cap t^\bullet$ and $|Val_{\alpha_{time'}}(\overline{t,p})| = 1$. So when transition t stops firing, it will produce only one token with interruptible resource and moves this token to a place $p \in P_{IC_j} \cap t^\bullet$. At this moment at most one transition $t \in T_{C_j}$ will be enabled just like at time 0. t_{SC_j} and $t \in T_{RC_j}$ are two special type transitions in T_{C_j}. When t_{SC_j} fires, it breaks up the options of current active node and lets himself to be an active node so the firing of t_{SC_j} doesn't change number of active nodes. At the moment t_{SC_j} or $t \in T_{RC_j}$ stops firing, they may recover the breakpoint $Bp_{c_j}(t_{SC_j})$ or $Bp_{c_j}(t)$. The breakpoint becomes an active node again. So the number of active nodes doesn't change. Therefore case 2 is immediate.

Definition 3.9: A finite occurrence sequence is a sequence of markings and modes:

$$M^*_{time_1}[t_1, \alpha_{time_1}\rangle M^*_{time_2}[t_2, \alpha_{time_2}\rangle \cdots M^*_{time_n}[t_n, \alpha_{time_n}\rangle M^*_{time_{n+1}}$$

Such that $n \in N$ and $M^*_{time_i}[t_i, \alpha_{time_i}\rangle M^*_{time_{i+1}}$ for all $i \in \{1, 2, 3, \cdots, n\}$, $M^*_{time_1}$ is the start Marking, $M^*_{time_{n+1}}$ is the end marking and n is the length.

Property analysis

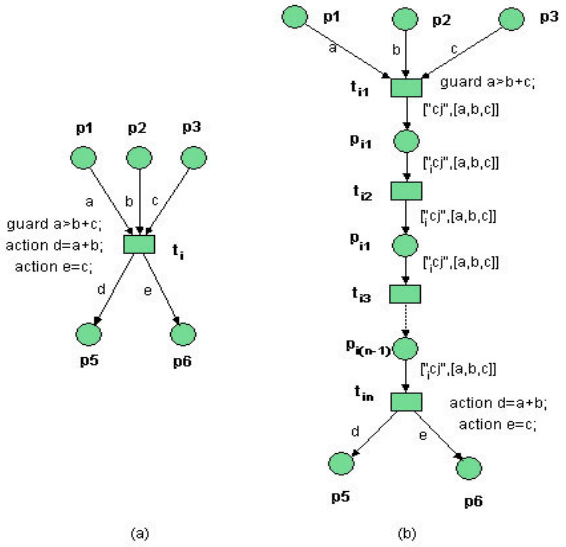

Fig. 3: An example of transform rule step 1

Here, we will define an equivalent relation between interruptible subnet of EHLTPN and $HLPN_{AT}$ with priority. Each interruptible subnet of EHLTPN $\forall N_{C_j} \in NC$, $N_{C_j} = (P_{C_j}, T_{C_j}, C_{C_j}, B_{C_j}, t_{SC_j}, T_{RC_j}, bp_{C_j})$ can be transformed into a behaviorally equivalent subnet of $HLPN_{AT}$ with priority, denoted by $\mathsf{HLPN}_{\mathsf{AT}} = (\mathsf{NG}, \mathsf{S}, \mathsf{C}, \mathsf{AN}, \mathsf{AC}, \mathsf{D}, \mathsf{P}_{ri}, \mathsf{M}_0^*)$, where $\mathsf{NG} = (\mathsf{P}, \mathsf{T}, \mathsf{F})$, $\mathsf{AN} = (\mathsf{A}, \mathsf{TC})$, $\mathsf{P}_{ri} : \mathsf{T} \to N$ is a function, annotated each transition with a natural number representing the priority level of each transition. Of two natural numbers, the bigger one has higher priority level.

The transform rule includes following steps:

Step 1: Construct $\mathsf{HLPN}_{\mathsf{AT}}$ and instead of $\forall t_i \in T_{IC_j}$ with a subnet which is composed of a serials of places and transitions.

(i). All the places of P_{C_j} belong to P and the types of these place keep no change. Thus $\forall p \in \mathsf{P}$ iff $\exists p \in P_{C_j}$ and $\forall p \in \mathsf{P}, \mathsf{C}(p) = C_{C_j}(p)$ iff $p \in P_{IC_j}$, $\mathsf{C}(p) = C(p)$ iff $p \in P_{NIC_j}$;

(ii). All the transitions which belong to T_{NIC_j} belong to T and the input arcs, output arcs of these transitions also belong to $\mathsf{HLPN}_{\mathsf{AT}}$. The guards, the actions and the initial remaining firing times annotated with these transitions and the annotations annotated with their input arcs and output arcs keep no change. Hence $\forall t \in \mathsf{T}$ iff $\exists t \in T_{NIC_j}$. For $\forall t \in \mathsf{T}$, following properties should be satisfied: $\forall p \in {}^\bullet t$, $(p,t) \in \mathsf{F}$ iff $\exists (p,t) \in F$, and for $\forall p \in {}^\bullet t$, $\mathsf{A}(p,t) = A(p,t)$; $\forall p \in t^\bullet$, $(t,p) \in \mathsf{F}$ iff $\exists (t,p) \in F$, and for $\forall p \in t^\bullet$, $\mathsf{A}(t,p) = A(t,p)$; $\mathsf{AC}(t) = AC(t)$; $\mathsf{D}(t) = D(t)$ and $\mathsf{TC}(t) = TC(t)$;

(iii). Transition t_i , $t_i \in T_{IC_j}$, are replaced by a serials of transitions and places, denoted by $t_{i1}, p_{i1}, t_{i2}, p_{i2}, \cdots, p_{i(n-1)}, t_{in}$, $n \in N$ and $t_{i1}, t_{i2}, \cdots, t_{in} \in \mathsf{T}$, $p_{i1}, p_{i2}, \cdots, p_{i(n-1)} \in \mathsf{P}$. They are connected by arcs in sequence. These arcs are all belong to F , so that, $\forall t_{im}$, $1 \le m \le n-1$, $t_{im} \in {}^\bullet p_{im} \wedge |{}^\bullet p_{im}| = 1$, $t_{i(m+1)} \in p_{im}^\bullet \wedge |p_{im}^\bullet| = 1$. The initial remaining firing times corresponding to $t_{i1}, t_{i2}, \cdots, t_{in}$ are denoted by $d_{i1}, d_{i2}, \cdots, d_{in}$ and they satisfy following equation $d_{i1} + d_{i2} + \cdots + d_{in} = d_i$. The number of n lies on the time precise of this net. Assume the number of n is big enough, so that, for $\forall d$, $0 < d < d_i$, exists a place p_m and $d_{i1} + d_{i2} + \cdots + d_{im} = d$, $1 \le m < n$.

(iv). Reconnect all the input arcs of t_i to t_{i1} and the annotations annotated with these arcs keep no change. That is for $\forall p \in {}^\bullet t_i$, $(p, t_{i1}) \in \mathsf{F}$ iff $\exists (p, t_i) \in F$ and for $\forall p \in {}^\bullet t_i$, $\mathsf{A}(p, t_{i1}) = A(p, t_i)$,

hence ${}^\bullet t_i = {}^\bullet t_{i1}$.

(v). Reconnect t_{in} with all the output arcs of t_i and the annotations annotated with these arcs keep no change. That is, for $\forall p \in t_i^\bullet$, $(t_{in}, p) \in \mathbf{F}$ iff $\exists (t_i, p) \in F$ and for $\forall p \in t_i^\bullet$, $\mathbf{A}(t_{in}, p) = A(t_i, p)$, hence, $t_i^\bullet = t_{in}^\bullet$.

(vi). Assume $\{var_1, var_2, \cdots, var_m\} = \bigcup_{p \in {}^\bullet t_i} Var(A(p, t_i))$ is the set of variables annotated with input arcs of t_i. Then assign the multiset of terms annotated with arcs, for $\forall (t_{ik}, p_{ik})$, $1 \le k \le n-1$, $\mathbf{A}(t_{ik}, p_{ik}) = (C_j, (var_1, var_2, \cdots, var_m))$. For $\forall (p_{ik}, t_{i(k+1)})$, $1 \le k \le n-1$, $\mathbf{A}(p_{ik}, t_{i(k+1)}) = (C_j, (var_1, var_2, \cdots, var_m))$.

(vii). Type the new added places. $\forall p_{im} \in \{p_{i1}, p_{i2}, \cdots, p_{i(n-1)}\}$, $\mathbf{C}(p_{im}) = H_{r_i C_j}$, $H_{r_i C_j} = \{(C_j, (val_1, val_2, \cdots, val_m)) \mid val_1 \in H_{var_1}, val_2 \in H_{var_2}, \cdots, val_m \in H_{var_m}\}$, H_{var_m} is the type of variable var_m.

(viii). Assign actions, $\forall t_{im} \in \{t_{i1}, t_{i2}, \cdots, t_{i(n-1)}\}$, $\mathbf{AC}(t_{im}) = \Phi$ and $\mathbf{AC}(t_{in}) = AC(t_i)$;

(ix). Assign guards, $\forall t_{im} \in \{t_{i2}, t_{i3}, \cdots, t_{in}\}$, $\mathbf{TC}(t_{im}) = true$ and $\mathbf{TC}(t_{i1}) = TC(t_i)$;

(x). For all $t \in T_{IC_j}$ except t_i do the same operations as (iii), (iv), (v), (vi), (vii), (viii) and (ix).

Step 2: Transform t_{SC_j}.

(i). BP is the set of all interruptible places after the implementation of step 1. $BP = P_{IC_j} \cup$ all the new added places in step 1 $= \{p \mid \mathbf{C}(p) = H_{rC_j}, r \in R\}$, is the set of interruptible places in $\mathbf{HLPN}_{\mathbf{AT}}$.

(ii). $Bp1 = \{bp \mid bp \in \{Val_\alpha(B_{C_j}(t_{SC_j}))\}, \alpha \in \mathbb{R}_{SC_j}\}$, is a finite set of all the possible resuming breakpoints of t_{SC_j} in subnet N_{C_j}, where $\mathbb{R}_{SC_j}$ is the set of all the modes which possibly enable t_{SC_j} when t_{SC_j} fires. Suppose the cardinality of $Bp1$ is $|Bp1| = n_R, n_R \ge 1$.

(iii). Transform all the breakpoints in $Bp1$ to $Bp1^*$, a finite set of all the possible resuming breakpoints of t_{SC_j} in $\mathbf{HLPN}_{\mathbf{AT}}$. All the breakpoints' type in $\mathbf{HLPN}_{\mathbf{AT}}$ are p. At first add all the breakpoints of type p in $Bp1$ to $Bp1^*$ directly. Then transform all the breakpoints of type t to type p and add them to $Bp1^*$. If $bp \in Bp1 \wedge bp.type() = p$, then $bp \in Bp1^*$. If $bp \in Bp1 \wedge bp.type() = t$, then find a place $p_{bp.ID()m}$ added in step1, $1 \le m \le n-1$ in the serial $p_{bp.ID()1}, p_{bp.ID()2}, \cdots, p_{bp.ID()(n-1)}$, so that $d_{bp.ID()m} + d_{bp.ID()(m+1)} + \cdots + d_{bp.ID()n} = bp.rt()$. Then transforms bp to bp^*, $bp^* \in Bp1^*$. Assign the value of bp^*, $bp^*.setID(bp.ID()m)$, $bp^*.settype(p)$. Let $\{var_1, var_2, \cdots, var_n\} = \bigcup_{p \in {}^\bullet t_{bp.ID()}} Var(A(p, t_{bp.ID()}))$ is the set of variables annotated with the input arcs of $t_{bp.ID()}$ then $bp^*.settoken((C_j, (bp.f(var_1), bp.f(var_2), \cdots, bp.f(var_n))))$. Obviously, $|Bp1| = |Bp1^*| = n_R$. The elements of $Bp1^*$ are denoted by

$Bp1^* = \{bp_1, bp_2, \cdots, bp_{n_R}\}$.

(iv). If place $p_i \in BP$ and $bp_k \in Bp1^*$ then add a new transition t_{ik} to T and connect it with all the input places and output places as t_{SC_j}, terms annotated with these arcs keep no change, the actions, the guard and the initial remaining firing time of t_{ik} is same as that of t_{SC_j}. Hence $t_{ik} \in \mathsf{T}$, $d_{ik} = d_{SC_j}$; for $\forall p \in {}^\bullet t_{SC_j}, (p, t_{ik}) \in \mathsf{F}$, iff $\exists (p, t_{SC_j}) \in F$, for $\forall p \in {}^\bullet t_{SC_j}, \mathsf{A}(p, t_{ik}) = A(p, t_{SC_j})$; for $\forall p \in t_{SC_j}^\bullet$ $(t_{ik}, p) \in \mathsf{F}$, iff $\exists (t_{SC_j}, p) \in F$, for $\forall p \in t_{SC_j}^\bullet$ $\mathsf{A}(t_{ik}, p) = A(t_{SC_j}, p)$; $\mathsf{AC}(t_{ik}) = AC(t_{SC_j})$, $\mathsf{TC}(t_{ik}) = TC(t_{SC_j})$. Add a new arc $(p_i, t_{ik}) \in \mathsf{F}$ and $\mathsf{A}(p_i, t_{ik}) = tokens$. Declare variable bp_{C_j} of BreakPoint type and variable $token$ of $C_{C_j}(p_i)$ type in the declarations of this net (just need declare one time of the same data type) and then add follow terms to the actions of t_{ik}, $bp_{C_j}.setID(i)$, $bp_{C_j}.settype(p)$, $bp_{C_j}.settoken(token)$. Then assign the guard of t_{ik}, $\mathsf{TC}(t_{ik}) = (B_{C_j}(t_{SC_j}).ID() == bp_k.ID()) \wedge TC(t_{SC_j})$; Add a new arc $(t_{ik}, p_{bp_k.ID()}) \in \mathsf{F}$ and $\mathsf{A}(t_{ik}, p_{bp_k.ID()}) = B_{C_j}(t_{SC_j}).token()$.

(v). For all $bp \in Bp1^*$ except bp_k do the same operations as (iv).

(vi). For all $p \in BP$ except p_i do the same operations as (iv) and (v);

(vii). Del t_{SC_j} and all of its input arcs and output arcs.

Step 3: Transform T_{RC_j}.

(i). If $p_i \in BP$, $t_{Rk} \in T_{RC_j}$ then add a new transition t_{ik} to T and connect it with all the input places and output places of t_{Rk}, terms annotated with these arcs keep no change and the initial remaining firing time is same as that of t_{Rk}. Hence, $t_{ik} \in \mathsf{T}$, $d_{ik} = d_{Rk}$; for $\forall p \in {}^\bullet t_{Rk}$, $(p, t_{ik}) \in \mathsf{F}$ iff, $\exists (p, t_{Rk}) \in F$, for $\forall p \in {}^\bullet t_{Rk}$, $\mathsf{A}(p, t_{ik}) = A(p, t_{Rk})$; for $\forall p \in t_{Rk}^\bullet$ $(t_{ik}, p) \in \mathsf{F}$ iff $\exists (t_{Rk}, p) \in F$, for $\forall p \in t_{Rk}^\bullet$, $\mathsf{A}(t_{ik}, p) = A(t_{Rk}, p)$. The actions of t_{ik} is same as that of t_{Rk}, $\mathsf{AC}(t_{ik}) = AC(t_{Rk})$. Then assign the guard of t_{ik}, $\mathsf{TC}(t_{ik}) = (B_{C_j}(t_{Rk}).ID() == i) \wedge TC(t_{Rk})$; Add a new arc $(t_{ik}, p_i) \in \mathsf{F}$ and $\mathsf{A}(t_{ik}, p_i) = B_{C_j}(t_{Rk}).token()$.

(ii). For all $p \in BP$ except p_i do the same operations as (i);

(viii). For all $t_R \in T_{RC_j}$ except t_{Rk} do the same operations as (i) and (ii).

(ix). Del $\forall t_R \in T_{RC_j}$ and all of their input arcs and output arcs.

Step 4: Set priority.

(i). All the transitions added in step 2 and step 3 have priority level 2.

(ii). All the other transitions have priority level 1 lower than level 2.

Figure 3-5 are three examples corresponding to transforming rule step 1, step 2 and step 3 separately. They are all composed of two subgraph (a) and (b). They are all transformed from subgraph (a) to subgraph (b). In Fig. 3, transition t_i in subgraph (a) is

transformed to a sequence of places and transitions, $t_{i1}p_{i1}t_{i2}p_{i2}\cdots p_{i(n-1)}t_{in}$. In Fig. 4, transition *t31* and *t32* are added in subgraph (b) to model the behaviors of t_{SC1} in subgraph (a) and they connect breakpoint *bp1* and *bp2* separately. In Fig. 5, transition *t31*, *t32*, *t41* and *t42* are added in subgraph (b) to model the behaviors of transition $tR1, tR2 \in T_{RC1}$ in subgraph (a).

The set of all markings and modes of N_{C_j} is denoted by M and Y. The set of all markings and modes of $\mathrm{HLPN}_{\mathrm{AT}}$ is denoted by M and Y. We use $M \mid P$ to denote the restriction of M to the subset of places specified by P. We use V to represent the set of variables which are added in the procedure of transform according the transform rule. Use $\mathsf{a}\backslash\mathsf{V}$ to denote the restriction of mode a where the subset of variables specified by V is discarded. All concepts written with ImprintMT Shadow font refer to $\mathrm{HLPN}_{\mathrm{AT}}$, while those written with normal font refer to N_{C_j} if doesn't declare specially.

In order to model the behaviors of subnet $N_{C_j} \in NC$ more practically, we divide each marking of N_{C_j} to two parts. One part is the marking in which all the tokens are produced by the transitions in N_{C_j}; Another part is the marking in which all the tokens are produced by the outside environment. They are denoted by $M_{time}^{C_j}$ and M_{time}^{env} separately, thus $M_{time} = M_{time}^{C_j} + M_{time}^{env}$.

Theorem 4.1: $\forall N_{C_j} \in NC$, each subnet $N_{C_j} = (P_{C_j}, T_{C_j}, C_{C_j}, B_{C_j}, t_{SC_j}, T_{RC_j}, bp_{C_j})$ can be transformed into a behaviourally equivalent subnet of $\mathrm{HLPN}_{\mathrm{AT}} = (\mathsf{NG}, \mathsf{S}, \mathsf{C}, \mathsf{AN}, \mathsf{AC}, \mathsf{D}, \mathsf{P}_{ri}, \mathsf{M}_0^*)$, where

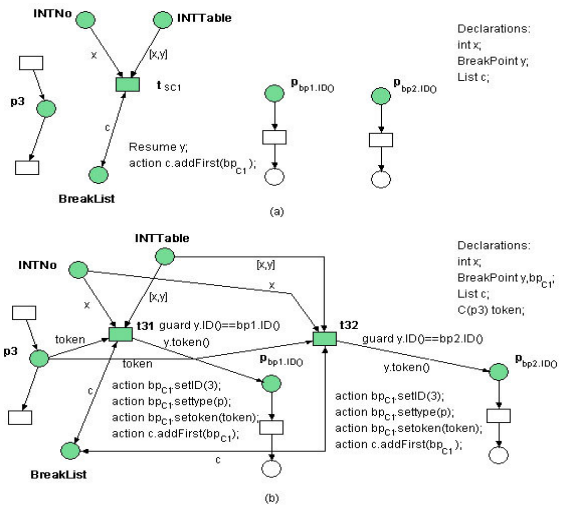

Fig. 4: An example of transform rule step 2

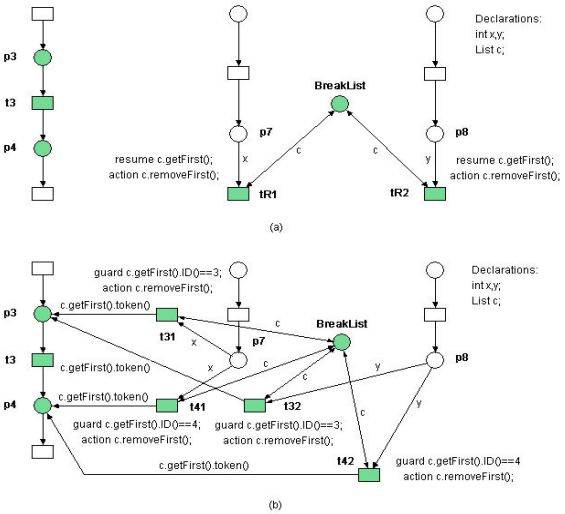

Fig. 5: An example of transform rule step 3

$\mathsf{NG} = (\mathsf{P}, \mathsf{T}, \mathsf{F})$, $\mathsf{AN}=(\mathsf{A}, \mathsf{T}, \mathsf{C})$. Then we have the following properties:

(i). $\forall M_{time} \in M, \exists \mathsf{M}_{time} \in \mathsf{M},\ M_{time} = (\mathsf{M}_{time} \mid P_{C_j}) \wedge M_0$

$= (\mathsf{M}_0 \mid P_{C_j})$, $time \geq 0$.

(ii). $\forall \alpha_{time} \in Y, \exists \mathsf{a}_{time} \in \mathsf{Y}, \alpha_{time} = \mathsf{a}_{time} \backslash \mathsf{V}$

(iii) $\forall M_{time_1}, M_{time_2} \in M, \forall \alpha_{time_1} \in Y, M_{time_1}[\alpha_{time_1}, t\rangle M_{time_2}$

$\Leftrightarrow \mathsf{M}_{time_1}[\mathsf{a}_{time_1}, t_1\rangle \mathsf{M}_{time_1+d_1}[\mathsf{a}_{time_1+d_1}, t_2\rangle \mathsf{M}_{time_1+d_1+d_2} \cdots$

$\mathsf{M}_{\substack{time_1+d_1+ \\ d_2+\cdots d_{n-1}}}[\mathsf{a}_{\substack{time_1+d_1+ \\ d_2+\cdots d_{n-1}}}, t_n\rangle \mathsf{M}_{time_2}$, $n \geq 1,\ n \in N$,

$time_1 + d_1 + d_2 + \cdots + d_n = time_2$

Proof

(i). Given that N_{C_j} and $\mathsf{HLPN}_{\mathsf{AT}}$ have the same environment, so that, at any time $M_{time}^{env} = (\mathsf{M}_{time}^{env} \mid P_{C_j})$, $time \geq 0$. Following transform rule step 1, step 2, step 3 and step 4 we know that transform rule doesn't change the initial marking of N_{C_j} and $\forall p \in P_{C_j}, p \in \mathsf{P}$, hence $M_0 = (\mathsf{M}_0 \mid P_{C_j})$. To prove $\forall M_{time} \in M$, $\exists \mathsf{M}_{time} \in \mathsf{M}$, $M_{time} = (\mathsf{M}_{time} \mid P_{C_j})$, for the inductive step, we need consider three cases:

Case 1: In the finite occurrence sequence of N_{C_j} doesn't exist t_{SC_j} or $t_R \in T_{RC_j}$. From transform rule step 1, we deduce that each transition $t \in T_{C_j}$ corresponding to a finite occurrence sequence of places and transitions of $\mathsf{HLPN}_{\mathsf{AT}}$, so that $\forall M_{time} \in M$, $\exists \mathsf{M}_{time} \in \mathsf{M}, M_{time} = (\mathsf{M}_{time} \mid P_{C_j})$.

Case 2: t_{SC_j} exists in the occurrence sequence of N_{C_j}. According transform rule step 2 and step 4, suppose t_{SC_j} corresponds to transition t in $\mathsf{HLPN}_{\mathsf{AT}}$. If t_{SC_j} occurs at time *time* when exist a place $p \in P_{IC_j} \wedge |M_{time}(p)| = 1$, transition t is enabled at the same time as t_{SC_j} and when it fires, it moves the same tokens from it's preset. So they have the same marking at that time. If t_{SC_j} occurs when exist a transition $t_1 \in T_{RunC_j} \wedge t \notin T_{NIC_j}$, t_{SC_j} will keep t_1's status in variable bp_{C_j} and sets its remaining firing time to 0. In $\mathsf{HLPN}_{\mathsf{AT}}$, transition t_1 is replaced by a sequence of transitions and places according transform rule step 1. At the moment when t_{SC_j} fires, exists a place in the sequence corresponding to the status of transition t_1 according transform rule step 2. When transition t fires it moves the same tokens as t_{SC_j} from its preset except the places which do not belong to N_{C_j}. After t_{SC_j} fires, if the type of $Bp_{c_j}(t_{SC_j})$ is p, then t and t_{SC_j} would produce the same marking. Otherwise, in N_{C_j} a transition would be resumed to "running" status and in $\mathsf{HLPN}_{\mathsf{AT}}$ a token would be moved to a place and from this place exists a sequence of places and transitions. The sum of these transition's initial firing time is same as the remaining firing time of the transition in N_{C_j}. They all have the same marking except the places that doesn't belong to N_{C_j}. The conclusion is obtained.

Case 3: $\exists t_R \in T_{RC_j}$ exists in the occurrence sequence of N_{C_j}. The proof is same as latter part of case 2.

(ii). This follows directly from transform rule step 1,2,3.

(iii). This follows directly from part (i).

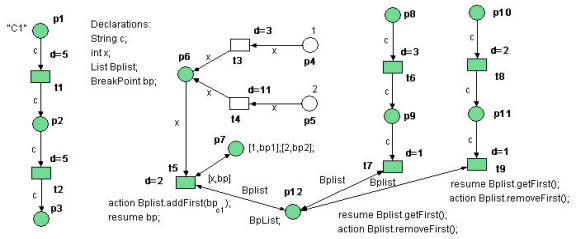

Fig. 6: An example of EHLTPN with two interrupt source

Example: Now, a simple example of EHLTPN is given. Seeing Fig. 6, this model is composed of two parts, one part models the interrupt requests from outside environment; the other part is an interruptible subnet of this model corresponding to interruptible resource processor C1. Place p4, p5 and transition t3, t4, model the interrupt requests from outside environment and they don't belong to N_{C1} .This part model has two interrupt sources, interrupt number 1 and interrupt number 2. They will be initialed at time 0 and will send interrupt requests at time 3 and time 11 to the subnet separately after the firing of transition t3 and t4. In the left is a model of a program section, it has resource C1 in place p1 representing that it has the right to run at time 0. p8, t6, p9 and p10, t8, p11 model two interrupt procedures. Place p6 is used to pend interrupt requests; Place p7 keeps the data of interrupt table. Suppose bp1 is a breakpoint at p8, bp2 is a breakpoint at p10. Place p12 maintains a FIFO list of breakpoints and at time 0 it is an empty list at time 0. Transition t5 is the Interrupt Switch Transition of N_{C1} . Transition t7 and t9 are two elements belong to T_{RC1} . The occurrence sequence of this model is:

$$M_0^*[\alpha_0, t1\rangle M_3^*[\alpha_3, t5\rangle M_5^*[\alpha_5, t6\rangle M_8^*[\alpha_8, t7\rangle M_9^*[\alpha_9, t1\rangle$$
$$M_{11}^*[\alpha_{11}, t5\rangle M_{13}^*[\alpha_{13}, t8\rangle M_{15}^*[\alpha_{15}, t9\rangle M_{16}^*[\alpha_{16}, t2\rangle M_{21}^*$$

Interrupts occur at time 3 and time 11. First time the breakpoint is transition t1, second time the breakpoint is place p2. After transition t3 fires, a token with number 1 is moved to place p6 representing an interrupt request of number 1 is pending. Thus transition t5 is enabled and then transition t5 fires and interrupts the firing of transition t1 at time 3. According the tokens moved from p6 and p7 to t5, transition t5 resumes the breakpoint of p8 and then runs the interrupt procedure p8, t6 and p9. At time 9, transition t7 resumes the firing of t1. Transition t1 continues its firing and at time 11 moves a token to p2. At the same time, an interrupt request of number 2 arrives and then transition t5 fires. Program section is interrupted at place p2. After the firing of t5, p10 is resumed and then runs the interrupt procedure p10, t8, p11. At time 16, transition t9 resumes breakpoint p2. Then transition t2 fires.

This model doesn't describe the function of interrupt controller, so interrupt number 1 and interrupt number 2 haven't priority level actually.

CONCLUSION

This study gives the formal definition of EHLPN by introducing interruptible subnet, actions and time to HLPN. EHLTPN includes interruptible subnets corresponding to different interruptible resource. In each subnet, an Interrupt Switch Transition and a set of Resuming Transitions are used to model the interrupt mechanism of this subnet. An example and a transform rule are also given in this study.

This new model cuts down the complexity of interrupt model of Petri nets. According this net, we can know when and where an interrupt occur and judge the affections produced by this interrupt. Basing on this model, models of interrupt controller and embedded operating system can be constructed directly and easily. The exact behaviors of distributed real-time embedded systems can be modeled freely in spite of the assumptions in other Petri nets about schedule, priority, firing time and so on.

Due to the limited of pages, the model of interrupt controller and embedded operating system are not given in this study, they will be given in another study.

REFERENCES

1. Wirth, N., 1977. Toward a discipline of real-time programming. Commun. ACM, 20: 8.
2. Budkowski, S. and P. Dembinski, 1987. An Introduction to ESTELLE: A Specification Language for Distributed Systems. Computer Networks and ISDN Systems, 14: 3-23.
3. Bolongesi, T. and E. Brinksma, 1987. Introduction to ISO specification language LOTOS. Computer Networks and ISDN Systems, 14: 25-59.
4. Saracco, R. and A.J. Tilanus, 1987. CCITT SDL: Overview of the Language and its Applications. Computer Networks and ISDN Systems, 14: 65-74.

5. Peterson, J.L., 1981. Petri Net Theory and the Modeling of Systems. Englewood Cliffs, NJ. Prentice-Hall.
6. Nissanke, N., 1997. Real-Time Systems. Prentice Hall. Englewood Cliffs, NJ.
7. Ramchandani, C., 1974. Analysis of asynchronous concurrent systems by timed Petri nets. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA (1974) Project MAC Report MAC-TR-120, MIT.
8. Merlin, P., 1974. A study of the recoverability of computer system. PhD. Thesis. Univ. California, Irvine, CA.
9. Ghezzi, C., D. Mandrioli, S. Morasca and M. Pezz`e, 1991. A Unified High-Level Petri-Net Formalism for Time-Critical Systems. IEEE Trans. Software Engg., 17: 160-172.
10. Mauricio Varea and Bashir AI-Hashimi, 2001. Dual transitions Petri Net based modeling technique for embedded systems specification. Design, Automation and Test in Europe. Conf. Exhibition 2001. Proc. 13-16 Mar., pp: 566-571.
11. Mauricio Varea and Bashir AI-Hashimi, 2002. Symbolic model checking of dual transition Petri Nets. Hardware/Software Codesign. CODES 2002. Proc. Tenth Intl. Symp. 6-8 May., pp: 43 – 48.
12. Miguel Felder, 2002. A Formal Design Notation for Real-Time Systems. ACM Trans. Software Engg. Methodol., 11: 149-190.
13. Giacomo Bucci andrea Fedeli, Luigi Sassoli and Enrico Vicario, 2004. Timed State Space Analysis of Real-Time Preemptive Systems. IEEE Trans. Software Engg., 30: 97-111.
14. Giacomo Bucci andrea Fedeli and Enrico Vicario, 2003. Modeling Flexible Real Time Systems with Preemptive Time Petri Nets. Proc. 15th Euromicro Conference on Real-Time Systems.
15. Zuberek, W.M., 1987. Modified m-timed petri nets in modeling and performance evaluation of system. Proc. 15th Ann. Conf. Computer Sci. St. Louis, Missouri, United States, pp: 261-268.
16. Christensen Søren and N.D. Hansen, 1993. Colored petri nets extended with place capacities, test arcs and inhibitor arcs. In: Ajmone Marsan, M.: Lecture Notes in Computer Science, Vol. 691; Application and Theory of Petri Nets 1993. Proc. 14th Intl. Conf., Chicago, Illinois, USA, pp: 186-205. Springer-Verlag.
17. Janusz Borkowski, 2002. Region-based Petri Nets for Modeling Interrupts and Cancellations. Parallel Computing in Electrical Engineering. PARELEC '02. Proc. Intl. Conf. 22-25 Sept., pp: 67-71.
18. Anonymous, 1997. High-Level Petri Nets-Concepts, Definitions and Graphical Notation. Committee Draft ISO/IEC 15909. Version 3.4.