Modeling Complex System Using T-subnet based Hierarchical Petri Nets

Zhijian Wang
Information Science School, Guangdong University of Business Studies, Guanzhou, China
Email: zjian@gdcc.edu.cn

Dingguo Wei
Information Science School, Guangdong University of Business Studies, Guanzhou, China
Email: weidg@scnu.edu.cn

***Abstract*—Net compression technologies are widely researched to avoid the problem of state explosion. Current researches of transition refinement and subnet abstraction mainly paid attention to preserve different attributes during the transformation, usually led to very strict conditions for the subnet, so that the application of these technologies was limited. Aiming to provide the same service and interface after transformation as original module while less restricts are given, a kind of transition subnet is put forward to model complex manufacturing system. The concept "standardized interface" is presented; transition subnets are classified into different types, the idea of "normalized subnet" is presented; Engineering Subnet is defined and a live and normalized subnet with finite live loopbacks is proved to be an Engineering Subnet. Because of live loopbacks owned by Engineering Subnets, the same interface and service as the original module are reserved after the transformation between an Engineering Subnet and the corresponding transition, mean while fewer preconditions are required for the Engineering Subnet compared with current researches.**

***Index Terms*—Petri nets, subnet, transition, interface, loopback**

I. INTRODUCTION

Petri net was presented by Carl Adam Petri in 1962[1], it is a modeling tool applicable to many systems, especially the discrete event dynamic systems [2]. Petri nets provide users an intuitive system describing method because of their graphical describing ability; as a mathematical tool, a Petri net model can be described by linear algebraic equations reflecting the behavior of the system. Although Petri nets have become an important modeling method today and are widely used, modeling complex systems with common Petri nets directly brings the problem of state explosion, system modeling and analyzing become extremely difficult and sometimes impossible. To overcome this problem, modularization and hierarchy are effective methods supplementing each other; usually modularization forms the basis of hierarchy.

The primary measures to realize hierarchy in Petri nets are transition refinement and subnet abstraction. Transition refinement is used in top-down system developing process, a subnet is designed to replace a transition and describe the function of that transition in detail; subnet abstraction is adopted in a bottom-up system developing process, a subnet is replaced by a transition so that the system model becomes more simple and understandable. Padberg presented a module concept for Petri nets, showed that the composition of modules preserves the property of safety[3]. Others researched the refinement, synthesis and condense of subnets or modules in Petri nets[4,5]. Hierarchical Petri nets are widely researched and applied, concepts such as reducible subnet, degree of subnet are defined by Lee[6].Hierarchical Petri nets have been used to model complex systems [7-8].

Same interface and service are required during the substitution between a transition and a subnet, and keeping same interface is the basic requirement of modularization, same outputs are required when same inputs are provided. Current researches stressed preserving properties such as boundedness, liveness and reversibility in the process of substitution [3-5]. In reference[8] a CO net for manufacturing systems was presented, CO net is a class of SISO Petri nets which is reversible, live, and keeps bounded; some refinement schemes were presented to apply Petri nets in complex flexible manufacturing systems [9,10].

For current researches there exists a conflict between the model equivalence and the modeling ability, usually too many strict restrictions are assigned to those subnets keeping good equivalence during the transformation[4-5,8-10], these restrictions limit the application of subnets greatly. As an example, a subnet is required to return to its initial marking after the firing of a transition sequence from input transition t_I to output transition t_O in reference [9], and the number of the tokens output by t_O must equal to that of what input by t_I. Such requirements are very difficult to be satisfied; in fact they are unnecessary in many occasions.

A reasonable assumption is that the subnet should return to a marking equal to its initially marking, the reached marking is not necessarily to be the initially marking directly; in some occasions even the repetitiveness is unnecessary in fact. In this paper a kind

of transition subnets is presented to solve above problem, the subnet can run in succession and is called Engineering Subnet.

This paper is arranged in the following way: section 2 investigates the interface and surrounding environments of subnets; section 3 presents the idea to normalize subnets, section 4 discusses the test nets and loopbacks of subnets, in section 5 two types of markings are researched, section 6 defines the concept of Engineering Subnet and analyzes its attributes, at last a few conclusions are given in section 7.

II. SUBNET AND INTERFACE

Definition 1: Given a P/T net $N=(P,T;F)$, if $P_S \subseteq P$, $T_S \subseteq T$, $F_S=((P_S \times T_S)\bigcup(T_S \times P_S)) \bigcap F$ and $SN=(P_S,T_S;F_S)$ is connected, then SN is the subnet of N.

Definition 1 originates in reference [12], but an extra condition is assigned that the subnet is required to be connected. This indicates the cohesion of a module; functions with no relationship are not allowed to be combined into the same module. Petri nets in this paper are connected P/T nets, namely the capacity function K is arbitrary but the weight function $W \equiv 1$, in the rest of this paper they are called nets in short.

Every element or subnet interacts with its environment in the complete model.

Definition 2: Given a net (or subnet) $N=(P,T;F)$, $x \in P \bigcup T$ and $Y \subseteq P \bigcup T$:

(1) $loc(x)=\{x\} \bigcup x^* \bigcup {}^*x$ is the local environment of x;

(2) ${}^*Y=\{y \mid \exists x \in Y,(y,x) \in F\}$;

(3) $Y^*=\{y \mid \exists x \in Y,(x,y) \in F\}$;

(4) $loc(Y)=\{Y\} \bigcup Y^* \bigcup {}^*Y$ is the local environment of Y;

(5) ${}^>Y={}^*Y-Y$;

(6) $Y^>=Y^*-Y$;

(7) $srd(Y)={}^>Y \bigcup Y^>$ is exterior environment of *Y*.

Item (1), (2) and (3) originate in reference [12], item (4) is an extended definition by this paper.

The local environment *loc(Y)* is composed by two parts: *Y* and those outside *Y*. If the elements of *Y* form a subnet, the above two parts should be considered separately in actual application, so the concept "exterior environment" of *Y* is presented. For a subnet *SN*, the exterior environment can also be called "exterior interface". ${}^>SN$ is the exterior input transitions and places of *SN* (exterior input interface), $SN^>$ is the output transitions and places of *SN*(exterior output interface), $srd(SN)$ is the elements in *N* which interact with *SN*.

Corresponding to exterior interface of subnets, inner interface or just interface in short, is defined in following definition 3.

Definition 3: Given a subnet $SN=(P_S,T_S;F_S)$, which is the subnet of net $N=(P,T;F)$:

(1) $SNI_P{}^- = \{p \mid p \in P_S \wedge ({}^*p - T_S \neq \phi)\}$ is the input place interface set of SN;

(2) $SNI_P{}^+ = \{p \mid p \in P_S \wedge (p^* - T_S \neq \phi)\}$ is the output place interface set of SN;

(3) $SNI_P{}^\# = SN_P{}^- \bigcup SN_P{}^+$ is the place interface set of SN;

(4) $SNI_T{}^- = \{t \mid t \in T_S \wedge ({}^*t - P_S \neq \phi)\}$ is the input transition interface set of SN;

(5) $SNI_T{}^+ = \{t \mid t \in T_S \wedge (t^* - P_S \neq \phi)\}$ is the output transition interface set of SN;

(6) $SNI_T{}^\# = SNI_T{}^- \bigcup SNI_T{}^+$ is the transition interface set of SN;

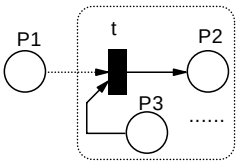
(7) $SNI^- = SNI_p{}^- \bigcup SNI_T{}^-$ is the input interface set of SN;

(8) $SNI^+ = SNI_p{}^+ \bigcup SNI_T{}^+$ is the output interface set of SN;

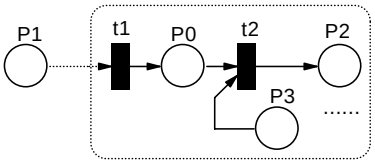
(9) $SNI^\# = SNI_P{}^\# \bigcup SNI_T{}^\#$ is the interface set of SN.

Although the concept "border" was presented in reference [12], it was used to describe a set of transitions, places and corresponding arcs, it is more similar to the subnet in this paper, but maybe not connected. We consider "border" is more suitable to describe the physical border of a net.

Definition 4: Given a subnet *SN*, if $(SNI^+)^* = SN^> \wedge {}^*(SNI^-) = {}^>SN$, then *SN* is termed to be interface standardized.



(a) A non-standardized interface



(b) The standardized interface

Figure 1. An example of how to standardize a non-standardized interface

Standardized interfaces help to make the interface between a subnet and its surrounding environments become clearer. Inputs of a transition in standardized interfaces decided completely by a subnet's surrounding environment and have nothing to do with other

restrictions inside the subnet. From the standpoint of modularization, this means the input parameters of a lower layer module are provided by some upper layer ones, what inside a module is to realize its function and provides service for upper layer modules, the lower layer module doesn't interfere the execution of the upper layer module. For the same reasons, a place in input interfaces works as a buffer of the module and accepts input variables; transitions inside the module decide how to use these variables, but they cannot influence the inputting. As to the input transition interfaces, the output transition or place interfaces, similar occasions come into existence. The interface in figure 1(a) is a non-standardized transition interface and it's standardized in figure 1(b).

III. TRANSITION SUBNET

Definition 5: For a subnet $SN=(P_S,T_S;F_S)$, if ${SN_P}^{\#}=\phi$ and ${SN_T}^{\#}\neq\phi$, namely $SN^{\#}={SN_T}^{\#}$,then SN is called a transition subnet(T-subnet) and:

(1)SN is a destination transition subnet(T^--subnet) if $SN^{\#}={SN_T}^{-}\neq\phi\,({SN_T}^{+}=\phi)$;

(2)SN is a source transition subnet(T^+-subnet) if $SN^{\#}={SN_T}^{+}\neq\phi\,({SN_T}^{-}=\phi)$;

(3)SN is a normal transition subnet($T^{\#}$-subnet) if ${SN_T}^{+}\neq\phi, {SN_T}^{-}\neq\phi$.

Transition subnets can be divided into several types according to their interfaces. Source subnets and destination subnets are named in allusion to their upper layer net models: if *SN* consumes tokens then it's a destination subnet; if *SN* produces tokens then it's a source subnet; if *SN* produces tokens after consuming tokens then it's a normal subnet.

According to the number of inputs(SNI^-) and outputs(SNI^+), subnets can be divided into different types as: ZI(Zero Input), SI(Single Input), MI(Multi Input), ZO(Zero Output), SO(Single Output) and MO (Multi Output). SISO $T^{\#}$-subnets are the most widely researched and applied T-subnets, what researched in reference [8] and [9] all belong to this type.

T-subnet abstraction is the process of replacing a T-subnet with a transition in order to simplify a complex Petri net model; it's an important method to realize hierarchy in Petri nets. According to reference [12], abstraction is divided into simple abstraction and strict abstraction, from this point of view, what researched in this paper is (the extension of) simple abstraction, because strict abstraction doesn't process subnet according to definition 1, it is in fact the fusion of transitions or places.

Above subnets and corresponding abstractions are listed in figure 2, what outside the dotted frame is the interface and what inside the dotted frame is the rest of the subnet.

Transition refinement is a reversed operation of T-subnet abstraction, which replaces a transition with a T-subnet. T-subnet abstraction is a bottom-up method and transition refinement is a top-down method.

Although the abstraction and refinement of net model operates reversely, they are not certainly mapped one to one. If a subnet *SN* is selected, the abstraction of *SN* is determined and no alternative exist. But for subnet refinement even the subnet *SN* and transition t_N are determined, there exists different schemes for the mapping transformation between the arcs of t_N and those of the subnet. According to the points of view in modularizing design, to define a module, only the inputs (${}^*t_N$), outputs ($t_N{}^*$) and module function (${}^*t_N \rightarrow t_N{}^*$) are required to be decided, the details of module realization is flexible.

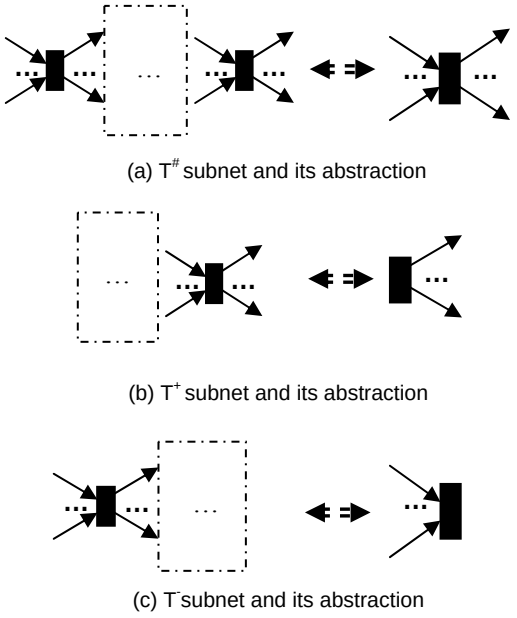


Figure 2. Typical T-subnets and their abstraction

So if the preconditions ${}^{>}SN={}^*t_N$ and $SN^{>}=t_N{}^*$ are satisfied, the structure of *SN* has different choices. According to firing rules of transitions, all inputs of a transition should be satisfied($\forall p\in {}^*t_N: M(p)>0$), and all outputs will be obtained at the same time ($\forall p\in t_N{}^*: M(p)=M(p)+1$), so the subnet *SN* should meet a condition, namely the execution of the subnet should consume one token from all places in ${}^{>}SN$ and produce one token for every place in $SN^{>}$.

Intuitively there exist different subnet structures meeting above condition under the same surrounding environment. For a normal transition t_N (${}^*t_N\neq\phi$ and $t_N{}^*\neq\phi$), the most intuitionistic method is to divide it into three parts as figure 2(a): the input part, the output part and the middle processing part. Guided by this idea, the concept of normalized subnet is presented in following definition 6.

Definition 6: A subnet *SN* is termed as normalized subnet if: (1) *SN* is an interface standardized subnet; (2) $|SNI^{-}| \leq 1$; (3) $|SNI^{+}| \leq 1$.

Intuitively all interface standardized SISO、SIZO、ZISO subnets are normalized subnets, and are named normalized SISO subnets, normalized SIZO subnets and normalized ZISO subnets respectively, as for T-subnets, they are normalized $T^{\#}$-subnets, normalized T^{-}-subnets and normalized T^{+}-subnet. Although current researches focus on SISO subnets, SIZO subnets and ZISO subnets are widely used in actual system modeling.

Normalized T-subnets are not the only type of subnets meeting previous conditions, but they are the qualified subnets with the simplest interface and are easy to be defined explicitly. So we use normalized T-subnets as the standard subnet structures in the process of T-subnets based refinement and abstraction, other T-subnets can be transformed into normalized subnets if certain preconditions are met.

Subnet normalization helps to make a transition and the corresponding T-subnet looks similar - they own the same interface to their surrounding environment; but for subnet transformation, "looks similar" is not enough, "acts similarly" is required – the model should work almost the same way before and after the transformation (because of the difference of state spaces, the transformed model is impossible to be completely same as the original model). To make models act similarly, the behaviors of subnets must be studied carefully, so the test net is required.

IV. LOOPBACK OF SUBNET

To analyze a subnet, we usually transform the subnet into a live, bounded and token conservative test net, that's the test net of subnet *SN*(labeled as $\overline{SN}$), this is especially important if we want to analyze characteristics such as transition delay when time is introduced in the net model. Almost all current researches focus on SISO subnet, in a SISO subnet a close route can be setup by adding a test transition or place to connect the input and output elements together as figure 3.

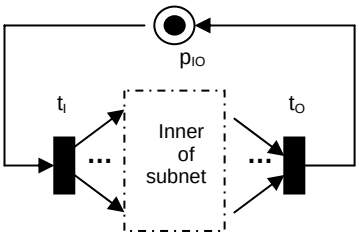


Figure 3. Test net of T#-subnet

Since the subnets researched in this paper are not limited to SISO subnets, we have to discuss the test net in detail. For a normal transition subnet in figure 3, following two places are required in order to run independently: an input place p_I working as the input of subnet from surrounding environment and an output place p_O working as the output of subnet to surrounding environment. In figure 3 we combine these two together as the place p_{IO}. Because a T^{+}-subnet has no outside input, so the time to fire t_O is decided by $\overline{SN}$ itself; similarly a T^{-}-subnet outputs nothing to surrounding environment, the subnet finishes an execution if t_I fires. Intuitively for a T^{+}-subnet or a T^{-}-subnet, p_{IO} in figure 3 can be deleted and that means its test nets is itself.

Definition 7: Given $\overline{SN}$ is the test net of subnet *SN*, if there exists a finite occurrence sequence $\sigma = M_1 t_1 M_2 t_2 M_3 ...$, t_I and t_O (if existed) occur in δ and occur only once, then $\tau = t_1 t_2 ...$ is called a loopback of $\overline{SN}$ (or *SN*). If there exists no loopback τ' and $\theta < \tau' < \tau$, then τ is a shortest loopback of $\overline{SN}$ (or *SN*).

For the three types of subnets, their loopbacks are in the following forms of expressions separately:

(1)$T^{\#}$-subnet: $\tau = \{\tau_1, t_I, \tau_2, t_O\}$;

(2)T^{+}-subnet: $\tau = \{\tau_1, t_O\}$;

(3)T^{-}-subnet: $\tau = \{\tau_1, t_I, \tau_2\}$.

where $\tau_1, \tau_2 = \{t_1, t_2 ... t_n\}$ are finite transition sequences and $t_1, t_2 ... t_n \in T_S - SNI_T^{-} - SNI_T^{+}$, that means except transitions in interface, the rest transitions inside the subnet can occur repetitively. The loopbacks set of *SN* is labeled as $Loop(SN)$.

The concept loopback is extended in definition 7, because for those ZI-subnet (T^{+}-subnet) or ZO-subnet (T^{-} subnet), their test net is not close, so it's impossible to connect their input and output so as to form a really close loopback. Also the subnet *SN* and its test net $\overline{SN}$ are not distinguished when loopbacks are referred, namely $Loop(SN) = Loop(\overline{SN})$, because they contain the same transition sequences for a same loopback.

V. MARKING SETS OF SUBNET

A loopback make sure the subnet can execute once from input to output.

Definition 8: Given $SM_1 \subseteq \{\overline{M0_S}\} \bigcup \overline{M0_S}[>$ (SM_1 is the subset of $\overline{SN}$'s initial marking and its reachable markings), if $\forall \overline{M_{S1}} \in SM_1 : \exists \overline{M_{S2}} \in SM_1$, $\exists \tau \in Loop(SN)$, $\overline{M_{S2}}[\tau > \overline{M_{S1}}$, then SM_1 is called the first type marking set of $\overline{SN}$, represented by $LR1(\overline{SN})$.

$LR1(\overline{SN})$ is the set of all markings reached by loopbacks from the initial marking $\overline{M0_S}$. Intuitively $\forall \overline{M_S} \in LR1(\overline{SN})$: $\overline{M_S}(p_{IO}) = 1$. So places outside subnets (p_{IO}) keep its original marking after the firing of a loopback τ. SN and $\overline{SN}$ are not distinguished in such occasion, namely $LR1(SN) = LR1(\overline{SN})$.

A normalized T-subnet with loopbacks can execute like a transition at least once under the initial marking a reasonable assumption is that the system can return to its original marking after one of its execution, or return to a marking equivalent to the original marking, this means the model should own the attribute of repetitiveness. Furthermore, even repetitiveness is not necessary in some systems. A door lock with engineering key works correctly in all states before and after using the switching key, but it can not return to the original state any longer if the switching is used.

According to the idea of modularization design, a subnet should be able to run continuously, namely in the new marking reached after an execution, the subnet can still run correctly when outside inputs are ready.

Definition 9: Given $SM1 \subseteq LR1(SN)$, if $\forall M_{S1} \in SM_1 : \exists M_{S2} \in SM_1$ and $\exists \tau \in Loop(SN)$, $M_{S1}[\tau > M_{S2}$, then SM_1 is a second type marking set of *SN*, represented by $LR2(SN)$; τ is called a live loopback.

A marking in $LR1(SN)$ may be "not live", although it is reached by a loopback, it's not sure that an other loopback can be founded under this state (it is not always a deadlock because there may still exist some transitions which can be fired); $LR2(SN)$ is a subset of $LR1(SN)$, every marking in $LR2(SN)$ can reach itself or an other marking in $LR2(SN)$ by a live loopback, so the markings in $LR2(SN)$ are "live", subnets in $LR2(SN)$ can run continuously .

Only those loopbacks with finite length make sense to a subnet, a finite length loopback indicates a repetitive routine composed by steady transition sequence and every such repetitive steady transition sequence maps with a processing routine in a manufacturing system. A non live loopback can be used only once, after that maybe the system reaches a marking where no any loopback can be found any longer. "maybe" is used here because there may exist such loopbacks that can be executed only once, but after that the system reaches a live state where other loopbacks are existed. So a "not live" loopback does not mean "dead", it just means a loopback can not be repeated.

A second type marking set of *SN* - $LR2(SN)$ can be divided into following different cases:

(1) $LR2(SN) = \varphi$.

In this case *SN* can not execute repeatedly and no live loopback exist. Such case is not researched any longer in this paper.

(2) $LR2(SN) = \{M0_S\}$, $\exists \tau \in Loop(SN)$, $M0_S[\tau > M0_S$.

This case is the presupposition of the research in reference [9] that *SN* returns to its initial marking after the firing of a loopback.

(3) $\exists M_{S1} \in M0_S[>, LR2(SN) = \{M_{S1}\}$, $\exists \tau \in Loop(SN)$, $M_{S1}[\tau > M_{S1}$.

This is the extension of case (2), the system enters a circular executing state in marking M_{S1} . Suppose $M0_S t_1 M_{Sk1} t_2 M_{Sk2} t_3 .....M_{S1}$, this can be a initialing process $(t_1, t_2, ...t_n)$ in advance in a real manufacturing system, after this the system begins a fixed operating routine τ to produce repeatedly. In case (2) we can regard that the system is initialized in the same way by $(t_1, t_2, ...t_n)$, so $(t_1, t_2, ...t_n)$ can be regarded as part of the processing routine. Of course *n=0* is possible, that means initialization is unnecessary in this system. In case (3) the system is required to be initialized only once (in certain time period).

(4) $M_{S1}, M_{S2}, M_{S3}... \in LR2(SN)$ and $M_{S1} \le M_{S2} \le M_{S3}...$, $LR2(SN)$ is an infinite set.

This is the case close to a recurrence system where a marking is covered by the markings reached from it. This case is not always to be a recurrence system, because the subnet is not always to be a live one since there may exist not live transitions. Intuitively unbounded places may exist in this case.

(5) $LR2(SN) = \bigcup SM_i$, and each SM_i form a marking set as case (3) or case (4); or forms similar system by the combination of several loopbacks.

This is the extension of case (3) or case (4).

(6) $|LR2(SN)| \ge 2$ and the conditions in case (5) are not satisfied.

In this case the markings in $LR2(SN)$ can be reached each other by loopbacks, that the system is revertible. In a real system usually $LR2(SN)$ is a finite set, so the live loopbacks in the system is finite.

VI. ENGINEERING SUBNET

According to the analysis above, the subnet which can be executed continuously is defined as following.

Definition 10: A normalized T-subnet SN is called an Engineering Subnet if:

(1) $\forall M_{S1} \in M0_S[>$, if $M_{S1}[t_1 >$ then t_1 is included in a loopback τ ;

(2) $Loop(SN) = LP_1 \bigcup LP_2$, where LP_1 is a non-empty, finite set of live loopbacks, LP_2 is a finite set of loopbacks; and $\forall \tau \in LP_2$: $M0_S[\tau > M_{S1} \wedge M_{S1} \in LR2(SN)$, or $\exists \tau_{11}, \tau_{12}...\tau_{21}, \tau_{22}... \in LP_2$, $M0_S[\tau_{11}, \tau_{12}...\tau, \tau_{21}, \tau_{22}... > M_{S1} \wedge M_{S1} \in LR2(SN)$.

Restrict (1) makes sure that the system will not reach a "not live" marking from the initial marking, a "not live" marking means in such a marking there exists no way to fire the output transition; restrict (2) makes sure that there exists no "dead" loopback, such loopbacks will bring the system into a "not live" marking. An Engineering Subnet *SN* is a subnet with the ability to execute continuously, it owns live loopbacks and the total number all loopbacks is finite; such a live loopback corresponds to a processing route, a not live loopback

corresponds to a initialing process, so the subnet will not trap into a "not live" marking.

Next we will analyze the relationship between subnet's liveness and its continuously executing ability. When we say a subnet *SN* is live we means its test net $\overline{SN}$ is live, because a subnet is not a complete net, whether its transitions can fire or not relies on its surrounding environment, so is not only decided by itself. If surrounding conditions is not satisfied, the subnet can not execute, but its test net is a complete net which can exist and execute independently.

Corollary 1: A live and normalized subnet owns and only owns live loopbacks.

Proof:

(1) Given a normalized T^+-subnet *SN*, since *SN* is live, there must exist a transition sequence τ_1 (t_O is not included in τ_1 , τ_1 may be empty), where $M0_S[\tau_1 > M_{S1}$, $M_{S1}[t_O > M_{S2}$, then $\tau = (\tau_1, t_O)$ forms a loopback and $M_{S2} \in LR1(SN)$, so *SN* owns loopbacks. $\forall M_{S1} \in LR1(SN)$, since SN is live, there must exist τ_1 (t_O is not included in τ_1), where $M_{S1}[\tau_1 > M_{S2}$, $M_{S2}[\tau_O > M_{S3}$, let $\tau = (\tau_1, t_O)$, then τ forms a loopback; above process can be repeated for M_{S3}, so τ is live, and $M_{S1} \in LR2(SN)$. In this way we can prove all elements in $LR1(SN)$ belong to $LR2(SN)$, and every loopback of SN is live.

(2)Similarly to above process, we can prove a normalized T^--subnet *SN* owns loopbacks and every loopback of *SN* is live.

(3)Given a normalized $T^{\#}$-subnet, since *SN* is live, there must exist transition sequences τ_1 and τ_2 (t_I and t_O are not included in τ_1 and τ_2 , τ_1 and τ_2 may be empty), where $M0_S[\tau_1 > M_{S1}$, $M_{S1}[t_I > M_{S2}$, $M_{S2}[\tau_2 > M_{S3}$ (since t_I in $\overline{SN}$ can not be fired again before t_O is fired, so the transition sequence τ_2 contains no t_I must be existed), $M_{S3}[t_O > M_{S4}$, then $\tau = (\tau_1, t_I, \tau_2, t_O)$ forms a loopback, then $M0_S[\tau > M_{S4}$, and so *SN* owns loopback; $\forall M_{S1} \in LR1(SN)$, since SN is live, there must exist transition sequences τ_1 and τ_2 (t_I and t_O are not included in τ_1 and τ_2), where $M_{S1}[\tau_1 > M_{S2}$, $M_{S2}[\tau_I > M_{S3}$, $M_{S3}[\tau_2 > M_{S4}$, $M_{S4}[\tau_O > M_{S5}$, let $\tau = (\tau_1, t_I, \tau_2, t_O)$, then τ forms a loopback; above process can be repeated for M_{S5}, so τ is live and $M_{S1} \in LR2(SN)$; In this way we can prove all elements in $LR1(SN)$ belong to $LR2(SN)$, and every loopback of *SN* is live.

With (1), (2) and (3) together, corollary 1 is proved.

According the proof process of above case (3) in corollary 1, and the properties of normalized $T^{\#}$-subnet, following corollary 2 is obtained.

Corollary 2: In a live and normalized $T^{\#}$-subnet SN, the firing of t_I is the sufficient and necessary condition to fire t_O.

So a subnet *SN* is not certainly to be a live one even every transition in *SN* is live when *SN* is treated as a part of the complete net model. t_3 in figure 4 can be live if it is a transition in a complete model, but we can find the subnet is not live by constructing its test net, the reason here is because the subnet doesn't satisfy the requirements in corollary 2.

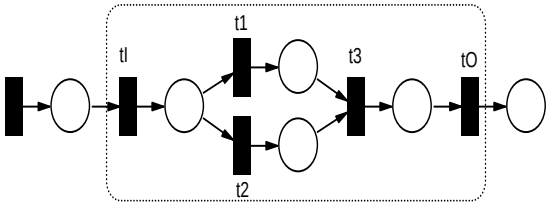


Figure 4. A not live subnet

Corollary 3: A live and normalized T-subnet with finite loopbacks is an Engineering Subnet.

Proof:

(1) We first prove if an arbitrary transition t_1 is fired, then t_1 is contained in a loopback.

$\forall M_{S1} \in M0_S[>$, suppose $M0_S[\tau_1 > M_{S1}$ (τ_1 is a transition sequence and may be empty). If $M_{S1}[t_1 > M_{S2}$, since *SN* is live, so:

a) For transition t_O in a T^+-subnet SN, there must exists transition sequence τ_2 (τ_2 contains no t_O), so that $M_{S2}[\tau_2 > M_{S3}$ and $M_{S3}[t_O >$; if t_O is not contained in τ_1 , according the specialties of normalized T^+-subnet, $\tau = (\tau_1, t_1, \tau_2, t_O)$ forms a loopback; if t_O is contained in τ_1 , let $\tau_1 = (\tau_{11}, t_O, \tau_{12}, \ t_O ... \tau_{1n-1}, t_O, \tau_{1n})$, then $\tau = (\tau_{1n}, t_1, \tau_2, t_O)$ forms a loopback.

b) For transition t_I in a T^--subnet SN, similarly to the process in a), we can prove if an arbitrary transition t_1 is fired, t_1 is included in a loopback τ ,

c) If SN is a $T^{\#}$-subnet, we have to discuss it in different cases.

If the firing of t_1 has nothing to do with t_I and t_O (τ_1 is unnecessary to contains t_I and t_O), then: if t_I and t_O are not contained in τ_1 , select a arbitrary loopback τ_2 , then $\tau = (\tau_1, t_1, \tau_2)$ forms a loopback and t_1 is contained in τ ; if t_I and t_O are included in τ_1 , let τ_1'' represents the transition sequence from the beginning to the last t_O in τ_1 (which is a loopback), τ_1' represents the transition sequence by take out τ_1'' from τ_1 , select an arbitrary loopback τ_2 , intuitively $M0_S[\tau_1'' > M_{S1}'[\tau_1' > M_{S1}$, then

$\tau = (\tau_1', t_1, \tau_2)$ forms a loopback and t_1 is contained in τ.

If the firing t_1 is associated with t_I and t_O, then at least one of t_I and t_O is included in τ_1, intuitively in τ_1 the number of t_O will not be greater than that of t_I (conclusion 1), because according to corollary 2, only one firing of t_I is required to fire t_O once; If t_I and t_O are required to fire many times so as to firing t_1, namely t_I and t_O occurred in τ_1 many times, since $\overline{SN}$ is live, so the firing of every pair of t_I and t_O has nothing to do with other t_I and t_O, so the transition sequence of τ_1 can be adjusted as $\tau_1' = (\tau_{11}, t_I, \tau_{12}, t_O, \quad \tau_{13}, t_I, \tau_{14}, t_O \ldots \tau_{1n-2}, t_I, \tau_{1n-1}, t_O, \tau_{1n+1}, t_I, \tau_{1n+2}, t_I \ldots)$, suppose after the firing of the last t_O the reached marking is M_{S1}', since $(\tau_{11}, t_I, \tau_{12}, t_O)$, $(\tau_{13}, t_I, \tau_{14}, t_O)$ … all form loopback separately, so $M_{S1}' \in LRl(SN)$, take out the loopbacks from τ_1', let $\tau_1'' = (\tau_{1n+1}, t_I, \tau_{1n+2}, t_I \ldots)$, then $M_{S1}'[\tau'' > M_{S1}$, and τ_1'' contains no t_O, it's composed by some t_I and other transitions(conclusion 2), to simplify the expressions, τ_1 is represented by τ_1'' in the rest of the proof; if t_I occurs in τ_1 m times and $m > 1$, since SN is live, then all t_I and t_O are live, so t_1 is at most the precondition of firing one t_O, it means for all t_O corresponding to the rest $(m-1)$ t_I, their firing don't require t_1 to be fired in advance, so in the M_{S1}, we can find transitions to form $m-1$ loopbacks before the firing of t_1, similar to the simplifying method used in the proof of conclusion 2, we change τ_1 into $\tau_1 = (\tau_L, \tau_{11}, t_I, \tau_{12})$, where τ_L is the $m-1$ loopbacks composed by t_I, t_O and corresponding transitions, suppose $M_{S1}[\tau_L > M_{S2}$, then $M_{S2} \in LRl(SN)$, let $\tau_1' = (\tau_{11}, t_I, \tau_{12})$ then $M_{S2}[\tau' > M_{S3}$, so $M_{S3}[t_1 >$ (t_1 is live in $\overline{SN}$, and loopbacks of $\overline{SN}$ can only fire in sequence, so the firing of $m-1$ t_O in advance will not affect t_1), suppose $M_{S3}[t_1 > M_{S4}$, because M_{S4} is reached from a first type marking M_{S2} by t_I firing once, so there must exists τ_2 (t_I and t_O are not included), so that $M_{S4}[\tau_2 > M_{S5}$, $M_{S5}[t_O >$,let $\tau = (\tau_1', t_1, \tau_2, t_O)$, then τ forms a loopback containing t_1.

So in case a), b) and c), if an arbitrary transition t_1 is fired, t_1 is included in a loopback τ, the restrict (1) in definition 10 is satisfied.

(2)The second restrict in definition is $Loop(SN) = LP_1 \bigcup LP_2$,according to preconditions, the number of loopbacks is finite, take corollary 1 into account, so LP_1 is a non-empty, finite set of live loopbacks, $LP_2 = \phi$.

With the proof of (1) and (2) together, Corollary 3 is proved.

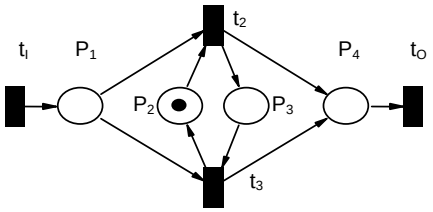


Figure 5. An example manufacturing system

An example of engineering subnet is the manufacturing system in figure 5. There exist two devices with completely same functions and performances, they can transform a rough part (the token in *p1*) into a finished product (the token in *p4*), where:

(1) t_I is the inputting transition to obtain a rough part into place p_1;

(2) p_2,p_3 are the tokens to operate these two devices;

(3) t_2 and t_3 represent the processing of the part p_1 by two same machines respectively.

(4) p_4 is the buffer of the finished products;

(5) t_O indicates the delivery of finished products.

This example is ordinary in real systems such as a dual server system. The model doesn't satisfy the requirements in reference[9], because the two markings M_1=*[x,1,0,y]* and M_2=*[x,0,1,y]* is different, but since the two machines work in the way, they make no difference to outside environment, the system always runs correctly when outside conditions are satisfied.

VII. CONCLUSION

Engineering Subnets aim to satisfy the requirements of hierarchical and modular modeling in a complex system but relatively relaxed restricts will be assigned. When Engineering Subnets are used to transform a model, the primary precondition is the attributes they owned, especially the existence of live loopbacks, the requirement of existing live loopbacks is relatively loose than that of current researches. The abstraction operation is to simplify the original model, in this case attributes of the net are relatively easy to be reserved; for system refinement, attributes of the net can be reserved under certain preconditions. Normalization of subnets solve the problem of making a subnet and the transition "look similarly", namely their interfaces to surrounding environments are the same; "live loopback" solves the problem of how to make a subnet and a transition "behave similarly".

If a subnet is required to be live, then every transition in the sub system should be live, but in an actual application system this is usually unnecessary. Maybe something is wrong in a module and results that some of its minor functions can not be executed successfully (corresponding transitions is not live), but the main functions still keep available, to surrounding environment the modular is still effective in such occasion. The precondition of Engineering Subnets is that the principal parts of a subnet are normal and the subnet will not

cumber the rest of the system, namely there still exists live loopbacks in its test net, so the modular can input and output normally.

ACKNOWLEDGMENT

This paper is based on "A Class of Petri Nets for Modular and Hierarchical System Modeling", which is appeared in proceedings of ICYCS2008.

Project supported by the Natural Science Foundation of Guangdong Province (Grant No. 8451032001001610), Natural Science Foundation of Guangdong Province (Grant No. 0630970).

REFERENCES

[1] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, 1989, vol. 77(4), pp. 541-580.

[2] J. H. Noguera, E. F. Watson, "Analyzing Throughput and Capacity of Multiproduct Batch Processes", *Journal of Manufacturing Systems* ,2004, vol. 23(3) , pp. 215-228.

[3] J. Padberg, "Safety Properties in Petri Net Modules", *Journal of Integrated Design & Process Science*, 2004, vol. 8(4), pp. 65 - 78.

[4] H. Huang, T. Y. Cheung, W. M. Mak, "Structure and behavior preservation by Petri-net-based refinements in system design", *Theoretical Computer Science*, 2004, vol. 328(3), pp.245−269.

[5] V. Khomenko, K. Alex, K. Maciej , "Merged processes: a new condensed representation of Petri net behaviour", *Acta Informatica*, 2006, vol.43(5), pp.307-330.

[6] K. H. Lee, J. Fabrel, "Hierarchical reduction method for analysis and decomposition of Petri Nets", *IEEE TRANS. SYST. MAN. CYBER*, 1985, SMC-15(2), pp. 272-280.

[7] M. Tittus, B. Lennartson, "Hierarchical supervisory control for batch processes", *IEEE Trans. on Control Systems Technology*, 1999, vol.7 (5), pp. 542-554.

[8] J. M. Proth, L. Wang, X. Xie, "A Class of Petri Nets for Manufacturing System Integration", *IEEE Transactions on Robotics and Automation*, 1997, vol. 13(3), pp.317-326.

[9] C. L. XIA, L. JIAO, W. M. LU, "Petri Net Refinement and Its Application in System Design", *Journal of Software*, 2006, vol.17(1), pp.11-19 (in Chinese).

[10] J. Padberg, M. Gajewsky, C. Ermel, "Rule-Based refinement of high-level nets preserving safety properties", *Science of Computer Programming*, 2001, vol.40, pp.97-118.

[11] C. Y. Yuan, *Petri Nets Theory and Application*, Beijing: Publishing House of Electronic Industry,2005.

[12] C. Girault, R. Valk, *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*, Berlin: Springer-Verlag, 2002.

Zhijian Wang was born in Changsha, Hunan Province, China in 1970. He obtained his Ph.D. degree in Control Theory and Control Engineering from Central South University in 2007 in China.

He is presently an associate professor of Computer Science in Information Science School, Guangdong University of Business Studies, Guangzhou, China. He is also the senior member of China Computer Federation. His current research interests include Petri nets, software engineering, and Computer Integrated Manufacturing Systems.

Dingguo Wei was born in Hunan Province, China in 1964. He obtained his Ph.D. degree in Computer Software and Theory from Fudan University in 2003 in China.

He is presently a professor of Computer Science in Information Science School, Guangdong University of Business Studies, Guangzhou, China. He is also the senior member of China Computer Federation. His current research interests include Petri nets, software engineering, and Electrical business.