Component-Level Reduction Rules for Time Petri Nets with Application in C2 Systems[1]

Jiacun Wang and Yi Deng

School of Computer Science
Florida International University
Miami, FL33199
{wangji, deng}@cs.fiu.edu

ABSTRACT

In this paper, we propose a set of component-level reduction rules for TPN's. Each of these reduction rules transforms a TPN component to a constant size of simple one while maintains the net's external observable timing properties. Consequently, our method works at a coarser level than that works in individual transition level, and fewer applications of our rules are needed to reduce the size of the TPN under analysis. We illustrate the use and benefits of our reduction rules by modeling and analyzing the response time of a command and control system to its external arriving messages.

1. INTRODUCTION

Time Petri nets (TPN's) have been used for the modeling and verification of various time dependent systems [1,2,3,6,7,9]. The analysis of a TPN is normally based on the enumerative technique developed by Berthomieu et al. [1,2]. For a complex or even middle-sized TPN, however, it is difficult to enumerate its reachable states, which is commonly referred as state-explosion problem. Sloan et al. developed several reduction rules for TPN analysis, which work at individual transition level [8]. These reduction rules help to reduce the complexity of TPN analysis to some extent. However, it is not a trivial work to automatically search the preconditions of applying these reduction rules for a complex TPN. In this paper, we propose a set of component-level reduction rules for TPN's (a component is a coarse grained subnet of a TPN). Each of our reduction rules transforms a TPN component to a constant size of simple one while maintains the net's external observable timing properties. Consequently, our method works at a much coarser level than those developed by Sloan et al., and fewer applications of our rules are needed to reduce the size of the TPN under analysis. The rest of the paper is arranged as follows: Section 2 informally introduces the concept of compositional TPN models, Section 3 proposes a set of component-level reduction rules, Section 4 addresses the analysis method for compositional TPN models, and Section 5 illustrates the use and benefits of our reduction rules by modeling and analyzing the response time of a command and control system to its external arriving message.

1. This work was supported in part by the NSF under Grant No. HDR-9707076, by Air Force Office of Scientific Research under Grant No. F49620-96-1-0221, and by NASA under Grand No. NAGW-4080. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official polices or endorsements either expressed or implied by the above named Agencies

2. OVERVIEW OF COMPOSITIONAL TIME PETRI NETS

The building blocks of a compositional TPN are *components* [6]. A component is a coarse grained subnet of a TPN. A compositional TPN model consists of two basic elements: *component TPN models* and *inter-component connections* (connections in brief). The component TPN models describe the real-time behavior and communication interface of the components; the connections specify how the components interact with each other and, in turn, form a composition model. All connections are defined using only communication interfaces, which gives us the flexibility to change the design of individual components without a need to void the analysis of the entire system.

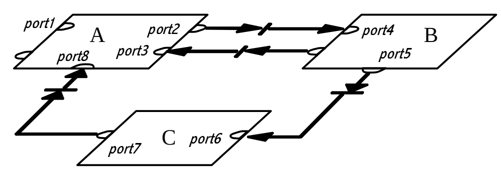


Figure 1. Framework of compositional TPN model.

Figure 1 shows an example of a compositional TPN model. The model has three components – *A*, *B*, and *C*. Each component model has two parts: (1) *communication ports* (denoted graphically by half circles), including *input ports* (e.g., *port*6) and *output ports* (e.g., *port*7), and (2) a TPN that describes the time-dependent, operational behavior of the component, that is, it defines the semantics associated with the ports. The communication between a component and its environment is solely through the ports. A connection represents a channel of interaction between components. It is modeled by a simple TPN and defines the direction of message flow and delay in the channel. For example, components *A* and *B* have a request-reply relationship that is modeled by the bi-directional channel. The communication is asynchronous message passing; that is, the sending component does not suspend itself if there is a concurrent activity available. Ports can also be divided into *external ports* (e.g. *port*1) and *internal ports* (e.g. *port*3 and *port*4). An internal port is defined to communicate with other components, while external ports are used to describe the inputs and outputs from and to the environment of the system.

3. COMPONENT LEVEL REDUCTION RULES

Before discussing the analysis method of compositional TPN models, we first introduce a set of component-level reduction rules for TPN's. These rules preserve the external observable timing properties of the components to be reduced, and contribute as the base of the analysis of compositional TPN models. Formally, we have

***Definition* 1:** Fix some rule for transforming a time Petri net N to a time Petri net N'. Let U be the set of transitions of N which are left completely unmodified by this transformation. We say that the transformation is *timing property preserving* if $L(N)|U = L(N')|U$.

In the rest of the paper, we denote the set of input ports and the set of output ports of component C as *C.PORT_IN* and *C.PORT_OUT*, respectively.

***Component-Level Reduction Rule* 1**

Let N be the TPN model of a system, and N_C the TPN model of a component in the system. *C.PORT_IN* = {*port*1}, *C.PORT_OUT* = {*port*2}. The component has no enabled transition under the initial marking of N. If

(1) whenever *port*1 receives a token, *port*2 is guaranteed to receive a token in the future, and

(2) *port*1 cannot receive another token until *port*2 has received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of two places: *port*1 and *port*2, and one transition: $t_{(1,2)}$, such that

(1) $port1^{*} = {}^{*}port2 = \{t_{(1,2)}\}$, ${}^{*}t_{(1,2)} = \{port1\}$, $t^{*}_{(1,2)} = \{port2\}$, while ${}^{*}port1$ and $port2^{*}$ remain unchanged, and

(2) $SI(t_{(1,2)}) = SI\_(port1, port2)$, the latter is the time delay interval for the token transfer from *port*1 to *port*2. (See Figure 2)

***Component-Level Reduction Rule* 2**:

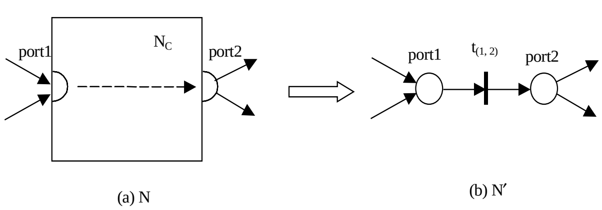


Figure 2. Illustration of reduction rule 1.

Let N be the TPN model of system S, and N_C the TPN model of component C of S. *C.PORT_IN* = {*port*1, *port*2}, *C.PORT_OUT* = {*port*3}, $port1^{*} = port2^{*}$. The component has no enabled transition under the initial marking of N. If

(1) whenever both *port*1 and *port*2 receive a token, *port*3 is guaranteed to receive a token in the future, and

(2) at least one of *port*1 and *port*2 cannot receive another token until *port*3 has received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of three places: *port*1, *port*2 and *port*3, and one transition: $t_{((1,2),3)}$, such that

(1) $port1^{*} = port2^{*} = {}^{*}port3 = \{t_{((1,2),3)}\}$, ${}^{*}t_{((1,2),3)} = \{port1, port2\}$, $t^{*}_{((1,2),3)} = \{port3\}$, while ${}^{*}port1$, ${}^{*}port2$ and $port3^{*}$ remain unchanged, and

(2) $SI(t_{((1,2),3)}) = SI\_((port1, port2), port3)$, the latter is the time delay interval from two tokens arriving in *port*1 and *port*2, respectively, to a token reaching *port*3. (See Figure 3)

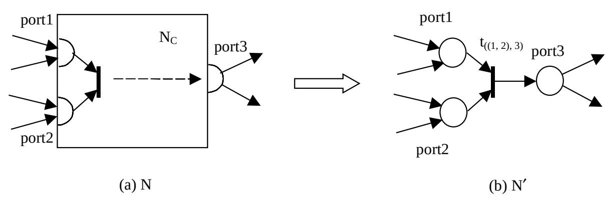


Figure 3. Illustration of reduction rule 2.

***Component-Level Reduction Rule* 3:**

Let N be the TPN model of a system S, and N_C the TPN model of component C of S. *C.PORT_IN* = {*port*1}, *C.PORT_OUT* = {*port*2, *port*3}. The component has no enabled transition under the initial marking of N. If

(1) whenever *port*1 receives a token, one and only one of *port*2 and *port*3 is guaranteed to receive a token in the future, and

(2) *port*1 cannot receive another token until one of *port*2 and *port*3 has received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of three places: *port*1, *port*2 and *port*3, and

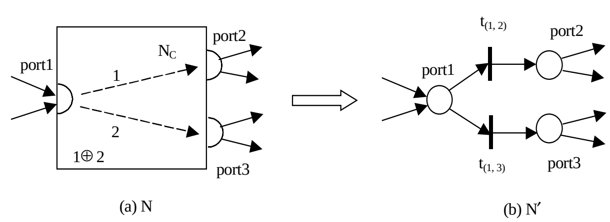


Figure 4. Illustration of reduction rule 3. The token reached *port*1 will either follows token flow path 1 to reach *port*2, or follow token flow path 2 to reach *port*3.

two transition: $t_{(1,2)}$ and $t_{(1,3)}$, such that

(1) $port1^* = \{t_{(1,2)}, t_{(1,3)}\}$, $^*port2 = \{t_{(1,2)}\}$, $^*port3 = \{t_{(1,3)}\}$, $^*t_{(1,2)} = {}^*t_{(1,3)} = \{port1\}$, $t^*_{(1,2)} = \{port2\}$, $t^*_{(1,2)} = \{port3\}$, while $^*port1$, $port2^*$ and $port3^*$ remain unchanged, and

(2) $SI(t_{(1,2)})$ = *SI*_(*port*1, *port*2), and $SI(t_{(1,3)})$ = *SI*_(*port*1, *port*3). (See Figure 4)

***Component-Level Reduction Rule* 4:**

Let N be the TPN model of a system S, and N_C the TPN model of component C of S. *C.PORT_IN* = {*port*1}, *C.PORT_OUT* = {*port*2, *port*3}. The component has no enabled transition under the initial marking of N. If

(1) whenever *port*1 receives a token, both of *port*2 and *port*3 are guaranteed to receive a token in the future, and

(2) *port*1 cannot receive another token until both of *port*2 and *port*3 have received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of four places: *port*11, *port*12, *port*2 and *port*3, and two transition: $t_{(11,2)}$ and $t_{(12,3)}$, such that

(1) $^*port11 = {}^*port12 = {}^*port1$, $port11^* = \{t_{(11,2)}\}$, $port12^* = \{t_{(12,3)}\}$, $^*t_{(11,2)} = \{port11\}$, $^*t_{(11,3)} = \{port12\}$, $t^*_{(11,2)} = \{port2\}$, $t^*_{(12,3)} = \{port3\}$, while $port2^*$ and $port3^*$ remain unchanged, and

(2) $SI(t_{(11,2)})$ = *SI*_(*port*1, *port*2), and $SI(t_{(12,3)})$ = *SI*_(*port*1, *port*3). (See Figure 5)

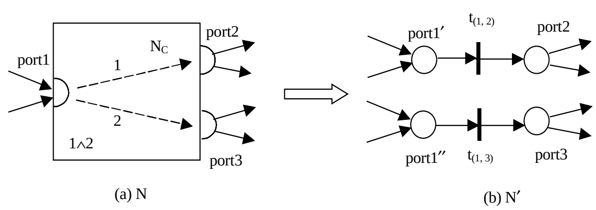


Figure 5. Illustration of reduction rule 4. The arrival of a token at *port*1 will result in a token reaches each of *port*2 and *port*3.

***Component-Level Reduction Rule* 5:**

Let N be the TPN model of a system S, and N_C the TPN model of component C of S. *C.PORT_IN* = {*port*1, *port*2}, *C.PORT_OUT* = {*port*3, *port*4}. The component has no enabled transition under the initial marking of N. If

(1) whenever both *port*1 and *port*2 receives a token, both of *port*3 and *port*4 are guaranteed to receive a token in the future, and

(2) at least one of *port*1 and *port*2 cannot receive another token until both of *port*3 and *port*4 have received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of four places: *port*1, *port*2, *port*3 and *port*4, and one transition: $t_{((1,2),(3,4))}$, such that

(1) $port1^* = port2^* = {}^*port3 = {}^*port4 = \{t_{((1,2),\,(3,4))}\}$, $^*t_{((1,2),\,(3,4))} = \{port1, port2\}$ and $t^*_{((1,2),\,(3,4))} = \{port3, port4\}$, while $^*port1$, $^*port2$, $port3^*$ and $port4^*$ remain unchanged, and

(2) $SI(t_{((1,2),\,(3,4))})$ = *SI*_((*port*1, *port*2), (*port*3, *port*4)). (See Figure 6).

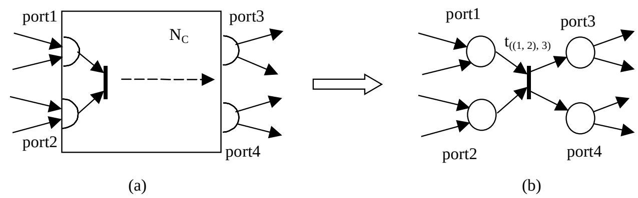


Figure 6. Illustration of reduction rule 5.

***Theorem* 1:** The component-level reduction rules 1-5 are timing property preserving [5].

4. ANANLYSIS OF COMPUTATIONAL TIME PETRI NETS

The modular nature of a compositional TPN model provides a natural support for incremental analysis. The algorithm works in two stages. In the first stage, we order the components so that the timing behavior of any component depends only on the components which are lower in this order. This stage assumes that the underlying dependency-graph of components is acyclic. In the next stage, we consider components one at a time in the order computed in the previous stage. Each component is verified and then reduced to a simple TPN of constant size that preserves the ports and all the externally observable time-dependent properties of the component.

The following definition is needed for the description of the analysis algorithm.

***Definition* 2:** Let *PORT_IN* and *PORT_OUT* be the input port set and output port set of a component, respectively. A pair of ports, <*port_in*, *port_out*>, where *port_in* ∈ *PORT_IN* and *port_out* ∈ *PORT_OUT*, is a *calling pair* of the component if a reply token from the environment is expected at *port_in* whenever a token is sent to some other component from *port_out*.

In Figure 1, for example, <*port*2, *port*3> is a calling pair.

The basic idea of the algorithm is to apply the component-level reduction rules introduced in Section 3 to replace each component by a very simple net of constant size that preserves all external observable time-critical properties of the component being replaced. Since the component-level reduction rules work at a much coarser level than the reduction rules given in [8], which work at individual transition level, consequently, fewer applications of the component-level reduction rules are needed to reduce the size of the model being analyzed.

Because of dependencies among components we need to be careful in choosing the order in which components are considered. For example, if a component has a calling pair, it needs to interact with other components. We say that component

C_i *depends on* component C_j, denoted by $C_i \rightarrow C_j$, if after C_i sends a token from the output port of its calling pair, it can not receive a token at the input port if we remove C_j away from the system. If $C_i \rightarrow C_j$ then C_j has to be analyzed before C_i. If $C_i \rightarrow C_j$ and $C_j \rightarrow C_l$, then C_l has to be analyzed before C_j, which, in turn, has to be analyzed before C_i. We can build a *dependency graph* such that there is a directed edge from component C_i to component C_j if and only if C_i depends on C_j. If this dependency graph contains a cycle, then we can't pick an order, so we assume that

***Assumption* 1:** The dependency graph does not contain a cycle.

In this case we can (linearly) order the component (e.g., by performing a topological sort [4]) in such a way that all dependency edges go from a component higher in this order to a component lower in the order. In other words, the first component does not depend on any other components so it can be verified and reduced to a simple TPN. The second component depends only on the first one, which has already been reduced, and so on. Suppose that there are *k* components, which are linked together by inter-related connection structures to form the system or subsystem. Now we can describe our incremental verification method as follows:

Algorithm

1. *For each component, list the reduction rule that can be applied to it*.
2. *Build the dependency graph on components and perform a topological sort. If the sorted order is C_1, C_2, …, C_k, then we know that $\forall\ i > j$, C_j doesn't depend on C_i.*
3. *Apply the proper reduction rule to each possible C_i from i =1 to k.*
4. *Compute the performance indices.*

5. MODELING AND ANALYSIS OF COMMAND AND CONTROL SYSTEMS

A Command and Control (C2) System is a huge and complex integration of distributed hardware and software components. A C2 system is a typical real-time system. Over delays in execution of any of its functions may degrade the kill probability, result in inefficient use of battle space, cause too many weapons to be assigned against the same target, and perhaps allow many targets to leak through a particular defense zone.

In this section, we illustrate the use of compositional TPN's to model and evaluate a tactic anti-air C2 system.

Typical structure of a tactic anti-air C2 system with two level command and control centers is shown in Figure 7, which consists of one first-level command center (indicated as *C2 center*) and three second-level command centers (indicated as *sub-centers*). A pair of (C2 center, sub-center) may be (division, regiment) or (brigade, battalion). They are geographically dispersed due to environmental and survivability reasons, which contributes to the distributed architecture of C2 organization.

We focus on the verification of requirements on the time delays in the execution of the system functions. Suppose that for a specific C2 system with the structure of Figure 7 there are following timing requirements:

(C1) The reaction time of the system, i.e., the time delay from an enemy intelligence message being received by any subcenter to a fire command agaisnt the enemy being issued by a corresponding fire unit, must be less or equal to 40 time units.

(C2) Since a C2 system is a closed-loop system, a constraint reflecting the time limitation for the feedback of damage assessment results should be included. This constraint is captured by the requirement that the time delay for a detail fire assignment scheme being made out by a subcenter to the result of the damage assessment referred to the execution of the scheme being received by the sub-center must be less or equal to 20 time units.

(C3) Since the bottleneck for information processing is often located in component C2 Center, the component is always asked to respond as soon as possible. This is captured by the requirement that the whole processing time for a group of messages from the three sub-centers must be less or equal to 22 time units.

All these requirements are applied under the assumption that no waiting time exists for any message at any processing stage. Notice that the timing data in this example is artificial and doesn't reflect actual data in a real system.

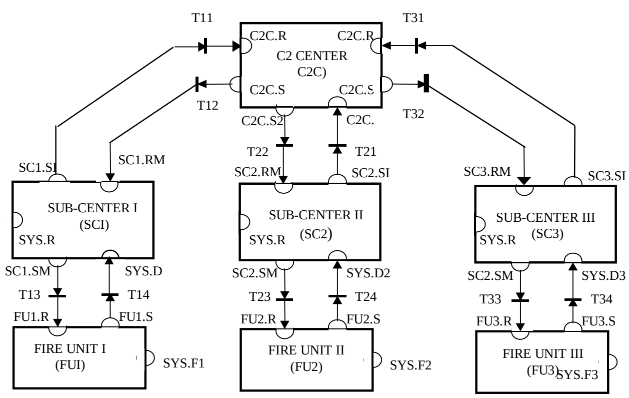


Figure 7. Composition interface of the C2 system.

5.1 COMPOSITIONAL TPN MODEL OF THE C2 SYSTEM

Figure 7 shows the interaction of system's components. From Figure 7 we know the system has seven components in total: a C2 Center (*C2C*), three Sub-Centers (*SC1*, *SC2*, *SC3*) and three Fire Units (*FU1*, *FU2*, *FU3*) (Air-radar groups are modeled as part of environment). Tables 1 and 2 show the meanings of all ports and all connection transitions. Here we assume that the three sub-centers have identical topology and timing properties and the three fire units have identical topology and timing properties. Because of the similarity, we only describe the ports and transitions for C2 Center, Sub-Center I, Fire Unit I and connections among them. Also assume that the three sub-centers always get external inputs at the same time.

Now we turn to the internal representation of each component. First let us consider the C2 Center. Its TPN model is shown in Figure 8. The C2 Center is composed of three *seats*: two *intelligence seats*, and *one decision-making seat*. The two intelligence seats communicate with the decision-making seat through a common memory. Its operational process is described as: The messages from three sub-centers are first copied and dispatched to the two intelligence seats (t1 fires). The two

Table 1. Legends of partial ports in Figure 7.

Port	Type	Description
SYS.R1	Input	A message from Air Radar Group I arrived
SYS.F1	Output	A combat command to Fire Unit I sent
SC1.SI	Output	Sub-Center I ready to send intelligence to Command and Control Center
SC1.RM	Input	Sub-Center I received command from Command and Control Center
SC1.SM	Output	Sub-Center I ready to send command to Fire Unit I
SC1.RI	Input	Sub-Center I received result of damage assessment from Fire Unit I
FU1.R	Input	Fire Unit I received command from Sub-Center I
FU1.S	Output	Fire Unit I ready to send result of damage assessment to Sub-Center I
C2C.R1	Input	Command and Control Center received message from Sub-Center I
C2C.S1	Output	Command and Control Center ready to send command to Sub-Center I

Table 2. Legends of transitions in Figure 7.

Transaction	Description	Firing Time
T11	Sub-Center I sends information to Command and Control Center	[1, 1]
T12	Command and Control center sends command to Sub-Center I	[1, 1]
T13	Sub-Center I sends command to Fire Unit I	[1, 1]
T14	Fire Unit I sends the result of loss assessment to Sub-Center I	[1, 1]

intelligence seats are responsible for performing *fusion* for these messages to achieve a relatively higher precision and overall situation figure, and then make *threatening assessment* independently for each target and sends the results to the decision-making seat (t2 and t3 fire). The decision-making seat works on a scheme of *battle planning* (t4 fires). The result is sent to the three sub-centers (T12, T22 and T32 fires). Notice that the required synchronization among messages from the three sub-centers has been modeled by transition t1, which does not fire until messages from all the three centers have been received.

Figure 9 shows the TPN model of component Sub-Center I. Each sub-center is composed of an intelligence seat and a decision-making seat. A sub-center's operational process is described as: The intelligence seat receives the message from its radar group and conducts *target discrimination*, *identification* and *tracking* (t5 fires), and further conducts *threatening assessment* (t6 fires), then sends the result to the C2 center (T11 fires). After receiving the scheme of *battle planning* from C2 center (SC1.RM is marked), the sub-center fuses it with related data in the database again so as to form a detailed scheme of *weapon-to-target assignment* (t7 fires). Furthermore, the results are sent to fire units (T13 fires).

As shown in Figure 10, the operational process of a fire unit is described as: When the scheme of *weapon-to-target assignment* arrives from its sub-center, it conducts *engagement control* (t8 fires) and sends fire command to weapons (t9 fires). Then, it conducts *damage assessment* (t10 fires), then feedback the assessment results to its corresponding sub-center in time (T14 fires).

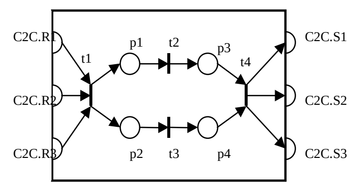


Figure 8 The TPN model of component *C2 Center.*

Firing times: t1:[1,2], t2:[3,5], t3:[3,5], t4:[5,6].

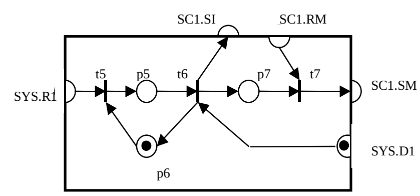


Figure 9. The TPN model of component *Sub-Center I.*

Firing times: t5:[2,3], t6:[1,2], t7:[4,6].

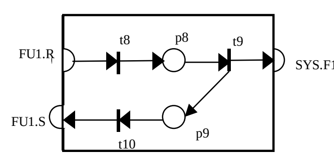


Figure 10. The TPN model of component *Fire Unit I*.

Firing times: t8:[4,6], t9:[1,2], t10:[5,7].

5.2 ANALYSIS OF THE C2 SYSTEM

We now illustrate the incremental verification on the C2 system. Since we have assumed that all requirements are applied under the assumption that no waiting time exists for any message at any processing stage, so our model for verification of these requirements, obtained by setting a token for each external input ports of Figure 7, satisfies Assumption 1, and thus our component-level reduction rules can be used. The C2 system consists of three types of components: C2 Center (C2C), three Sub-Centers (SC1, SC2 and SC3) and three Fire Units (FU1, FU2 and FU3). There are 6 dependency edges: SCi → C2C, i = 1, 2, 3, and SCi → FUi, i = 1, 2, 3, so we can order these components as

C2C, FU1, FU2, FU3, SC1, SC2, SC3.

Applying component-level reduction rule 5 and 4 to component C2 Center and Fire Unit I results in Figure 11(a) and 11(b), respectively. Applying enumerative analysis method as given in Section 4, we conclude

$SI(t_{(C2C.R,\ C2C.S)}) = [16, 20]$,

$SI(t_{(FU1.R,\ SYS.R1)}) = [5, 8]$, and

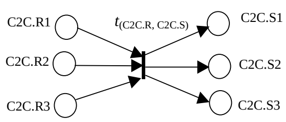
$SI(t_{(FU1.R,\ FU1.S)}) = [10, 15]$.

So the constraint (C3) is verified. Next, we consider the three Sub-Centers. Because we have assumed that the three sub-centers have identical topology and timing properties, the three fire units have identical topology and timing properties, and the three sub-centers always get external inputs at the same time, so we can use Figure 11(c), which shows Sub-Center 1 and its interaction with simplified C2 Center and Fire Unit 1, to verify constraints (C1) and (C2). Based on Figure 11(c), we obtain

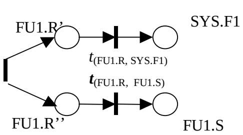
*D*_(SYS.R1, SYS1.F1) = [31, 42], and

*D*_(SC1.SM, SC1.RI) = [12, 17],

where *D*_(SYS.R1, SYS1.F1) represents the message transferring time delay between ports SYS.R1 and SYS1.F1, so



(a)



(b)

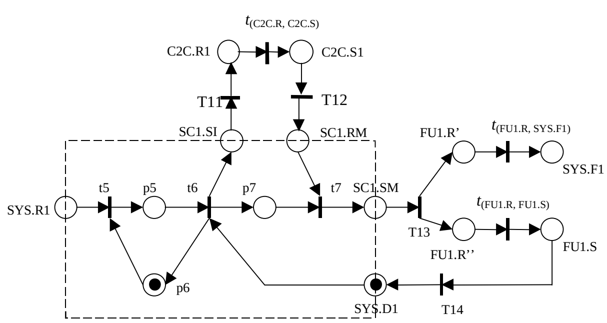


Figure 11. (a) Reduction of component C2 Center.
(b) Reduction of component Fire Unit 1 (Fire Unit 2, 3 have the same topology).
(c) The simple but equivalent model of the system.

does for *D*_(SC1.SM, SC1.RI). These values imply that (C1) and (C2) are satisfied. Therefore, all constraints have been verified based on the top-level model of the C2 system.

6. CONCLUSION

A set of component-level reduction rules for TPN is introduced. Each of these reduction rules transforms a TPN component to a constant size of simple one while maintains the net's external observable timing properties. Since these reduction rules work at a coarser level than that works in individual transition level, and fewer applications of our rules are needed to reduce the size of the TPN under analysis. The modeling and analysis of the response time of a command and control system to its external arriving messages are illustrated to show the use of these reduction rules.

REFERENCES

[1] B. Berthomieu and D. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transaction on Software Engineering*, 17(3), 259-275, 1991.

[2] B. Berthomieu, M. Menasche. An enumerative approach for analyzing time Petri nets, *Procedings of IFIP Congress*, Sept. 1983.

[3] G. Bucci and E. Member, Compositional validation of time-critical systems using communicating time Petri nets. *IEEE Transactions on Software Engineering*, 21(12), 969-992, 1995.

[4] T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms*, The MIT Press, 1994

[5] Y. Deng, J. Wang and R. K. Sinha. Integrated Architectural Modeling of Real-Time Concurrent Systems with Applications in FMS. *Technical Report, School of Computer Science, Florida International University*, 1997.

[6] Y. Deng, J. Wang, and R. Sinha, Integrated architecture modeling of real-time concurrent systems with applications in FMS, Proceedings of the 10th International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, USA June 18-20, 1998

[7] P. Merlin and D. Farber, Recoverability of communication protocols - Implication of a theoretical study, *IEEE Transactions on Communications*, 1036-1043, Sept. 1976.

[8] R. Sloan, U. Buy. Reduction rules for time Petri nets. *Acta Informatica*. 33, 687-706, 1996.

[9] J. Tsai, S. Yang, and Y. Chang, timing constraint Petri nets and their application to schedulability analysis of real-time system specifications, *IEEE Transactions on Software Engineering*, 21(1), 32-49, 1995.