

Performance Prediction Model Generator Powered by Occurrence Graph Analyzer of Design/CPN*

Insub Shin and Alexander H. Levis

System Architectures Laboratory
School of Information Technology and Engineering
George Mason University, MS 4D2
Fairfax, VA 22030, USA
E-mail: {ishin,alevis}@gmu.edu

ABSTRACT

This paper describes a methodology for generating a performance prediction model so that both qualitative (logical correctness) and quantitative (timeliness) properties of a real-time system can be evaluated. The methodology uses two types of tools: the reachability analysis tool of a Petri net and a discrete event-driven simulation technique of a queueing net. Typically, the complete specification of a complex system is obtained through a combination of multiple architectural perspectives. This paper focuses on the synthesis of a functional architecture and a physical architecture. Given that the logical behavior of a real-time system is described as a functional architecture, the logical property is verified using the Occurrence Graph Analyzer of Design/CPN [1]. Then, the timing property of the system, running on a communication network, is measured by simulation using NS (Network Simulator [2]). During the simulation, the partially ordered messages from the logical model are processed in the total order in the network traffic-scheduling scheme. The uniqueness of this methodology is the “off-line” simulation of both models, rather than “run-time” simulation. The techniques developed offer a means to use the Petri net properties for analysis. This is not possible in run-time simulation due to the open interfaces between the two models. This methodology provides the strengths of both a formal method and simulation techniques for performance evaluation of a real time system. The paper includes a description of the automated algorithms that transform a logical model (i.e., untimed Petri net model) of a real-time system into a performance prediction model (i.e., timed Petri net model).

1. INTRODUCTION

1.1 Motivation

In the systems engineering approach, a mission is decomposed into tasks, and resources are allocated (e.g., sensors, human decision makers, computing processors, communications networks, and weapon systems, etc.) to perform the task. The mission is structured as a set of tasks connected by synchronous/asynchronous channels and accomplished by the execution of tasks in series and/or in parallel. The execution of a task is constrained by external stimuli such as higher authority’s mission requirements or by the capabilities of assigned resources. For a time critical C3 (command and control, communication) system, analysis of the timing of each task and time management in a series of task executions are essential to guarantee that the system responds at the right time, or before but not after a due time.

* This work was supported in part by the Air Force Office of Scientific Research under Grant No. F49620-98-1-0179

In general, a real time system is characterized by its timely response to external stimuli [3]. A response consists of a series of task executions. A task execution is usually characterized by its start-time, execution-time, and deadline [4]. Timing constraints are then postulated between the external stimuli and the response. Those constraints express properties of end-to-end timing propagation delay. Maintaining functionally correct end-to-end values may involve a large set of interacting components. To ensure that the end-to-end constraints are satisfied, each of these components will be subject to its own intermediate timing constraints [5]. For a distributed decision-making organization dealing with time critical missions, the estimates of the elapsed time of each interacting component and the message delay can play a critical role in developing combat engagement rules, establishing decision thresholds, and battle time management.

At the early stage of the system development cycle, however, there are significant uncertainties in the performance parameters, task execution time, and network connection delays. They are due to incomplete system specifications and/or complexity of the problem. If the physical communication link between tasks is simply an immediately adjacent transmission link, the modeling effort may be easy. When multiple connections share common communications network resources other than single, immediate, adjacent transmission link, the cost of modeling is very high. In some cases, modeling the contention of communication resources may be practically impossible. The performance model for time critical missions must be well formalized. It also requires accurate timing estimate methods.

1.2 Approach

A C3 system for a high level military organization is a system-of-systems consisting of heterogeneous subsystems operating in distributed operational environments. The property or behavior of a system is specified and expressed in multiple perspectives or views. Figure 1 shows an example of those architectures.

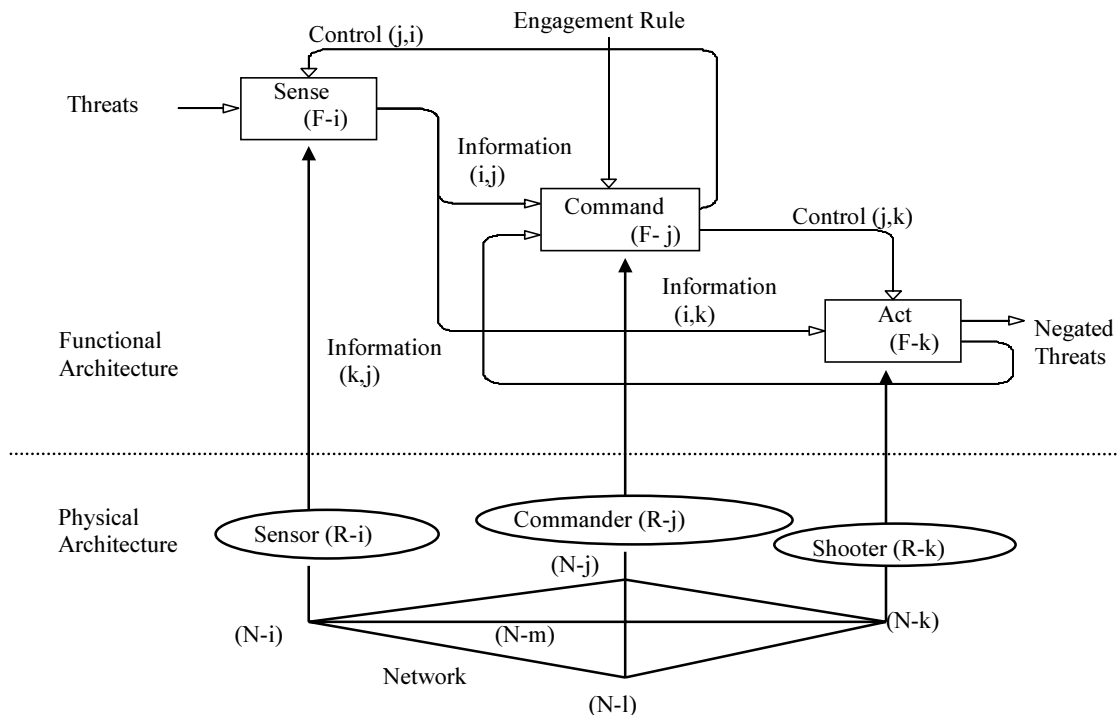


Figure 1. An Example of System Architecture

The logical behavior (e.g., computational correctness or control structure) of a system is specified in a functional architecture and the supporting resources are specified in a physical architecture. Predicting the performance of the system requires synthesis of the multiple views. Both functional and physical architectures describe or specify the same system. So they are not separate systems in a global sense. In this methodology, however, both architectures are considered as separate state machines. Then, the performance parameters are obtained by exchanging the necessary state information between two machines.

Developing an executable model is a way to predict the performance of a real time system. Operational formalisms [6,7] such as Petri nets and State Machines are suitable to specify and express executable models. One of the most important principles of military doctrine is synchronization. Using the Petri net formalism, the synchronization problem is resolved by its structure. In this methodology, the target performance prediction model is developed as a Timed Petri net. However, the methodology does not directly model the communications demands between tasks within the Petri net. The time values are estimated by simulations of a separate communications network model using realistic traffic messages that are generated in partial order from an untimed Petri net. We use the occurrence graph of a Petri net to generate those messages. While simulating the communication network model, the partially ordered messages shall be processed in a total order in its traffic-scheduling scheme.

1.3 Scope of Work

Finding all sequences of message passing from a full occurrence graph is very costly since the full occurrence graph contains all the reachable markings. However, if the functional architecture is specified as a deterministic conflict-free Petri net, the net has a single final state after firing all enabled transitions given initial markings. Also the markings of the net after firing a set of transitions are same regardless of the sequence of firings. So, if we can develop a functional architecture as a conflict-free Petri net, a partial occurrence graph with one sequence of firings can provide the order of message passing in the system.

Once a conflict-free Petri net is created using Design/CPN, the tool developed generates the whole state space information automatically. Next, the network topology generation tool is invoked to build the physical architecture. Once the physical architecture (communication network model) is created, the Network Simulator reads the state information, runs the simulation, and generates delay values. Finally, these values are introduced as an arc inscription to express the time delays in the Timed Petri net (performance prediction model).

The methodology has been implemented under the assumption that the system is an information system with a conflict free net structure. But the methodology and tools developed are neither limited to information systems nor conflict-free nets. If the Petri net is not conflict-free and if there exist multiple final states, the performance parameters will be obtained by repeated simulation. The tool developed can compute and select either a shortest path or a longest path from initial state to final states.

2. A METHODOLOGY TO DEVELOP A PERFORMANCE PREDICTION MODEL

The performance prediction model is generated using 6 major steps:

- Develop executable functional architecture
- Identify communications service access interfaces
- Create message dependency relations
- Simulate communications network
- Estimate network delays
- Introduce time to the executable functional architecture.

2.1 Step 1: Develop Executable Functional Architecture

The first step in the development of a functional architecture is the functional decomposition. The functional decomposition can be carried to any level of detail. It may depend on the level of abstraction and modeling purpose. In practical terms, it is carried out to the point that the lowest level tasks must be executed by a single resource [8]. To estimate the network delays, we need to identify the physical resources in a communications network architecture. So, the decomposition process may stop at the level at which the physical resources can be identified at the communication network and its elemental performance parameters are known.

Suppose that the functional architecture in Figure 1 describes a system to be used to clear threats in which a man-machine decision making system makes decisions based on input sensor values to respond to the threats. A corresponding Petri net structure can be depicted as shown in Figure 2.

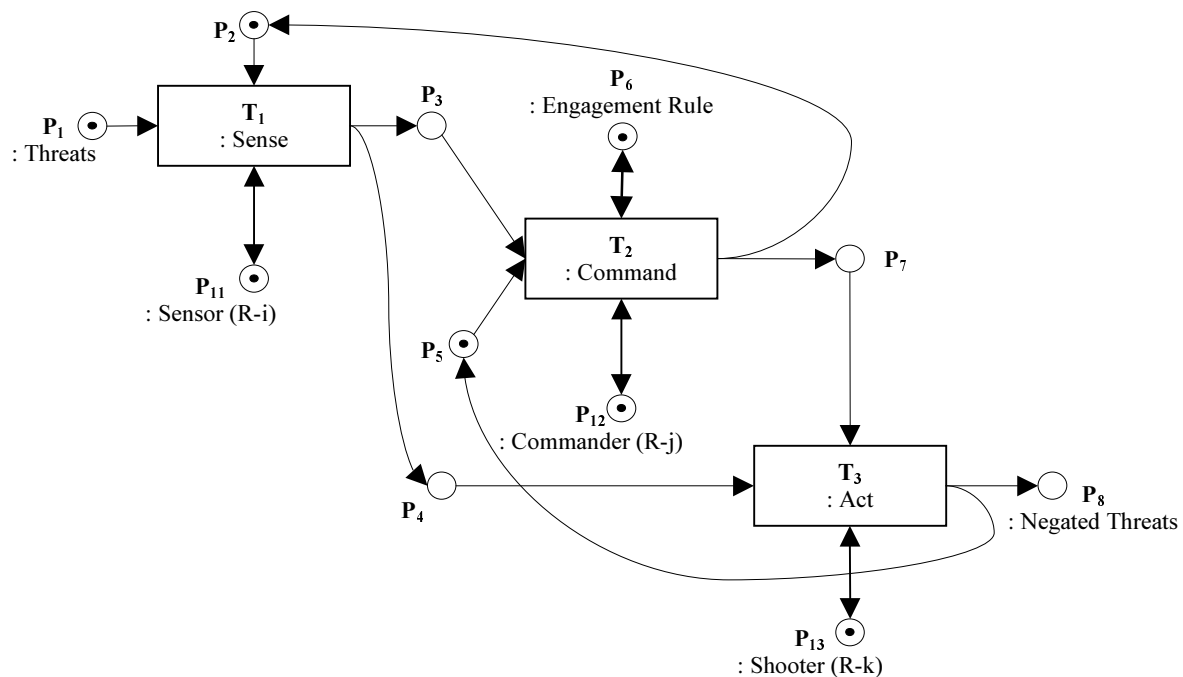


Figure 2. An Example of Executable Functional Architecture

Once a set of tasks is represented by "transitions", the allocation of resources to perform the tasks is represented by tokens at "places". Places P11, P12, and P13 in Figure 2 are used for assignment of resources to tasks. Resources R-i, R-j, and R-k in Figure 1 are instances of resources. These physical resources will have dual roles: one role as computing devices in the functional architecture and the other role as traffic generators in the communications network architecture. In Figure 1, Resources R-i, R-j, and R-k are the physical resources that play the dual roles. As a computing device, the physical resource can be interpreted as a server to carry out a task. Adding these physical resources to the executable functional architecture denotes that the task uses the service of the resource to execute the task. If those resources at places P11, P12, and P13, are used only one time and consumed, those thick arcs in Figure 2 will be uni-directed arcs from P11 to T1, P12 to T2, and P13 to T2. Similarly, if the engagement rule at place P6 is used only one time and consumed, the thick arc from P6 to T2 will be uni-directed. If the rule is used repeatedly and updated dynamically, the arc must express self-loop. Also the rule must have a time stamp and hence the color set of place P6 must be timed.

2.2 Step 2: Identify Communications Service Access Interfaces

The communication network simulation model can be generalized as a network of a set of nodes, namely N_1, N_2, \dots, N_n , and a set of links which are node pairs (i,j) between node i and node j . Each link has a capacity $C_{i,j}$, counted in bandwidth units. Links are undirected; (i,j) and (j,i) denote the same link. In this methodology, the communication network consists of four layers: Physical Layer, Data Link Layer, Network Layer, and all the Higher Layers [9]. The Higher Layers layer generates the application traffic, and the other 3 layers process this traffic. The message traffic has 7 attributes; message identification (ID) number, source node, destination node, message size, priority, protocol, and message dependency relation. The physical resources allocated to the functional architecture model generate the application network traffic. The same physical resources connected to a communication network model as its leaf nodes can be used to identify the communication service access points in the network as shown in Figure 3.

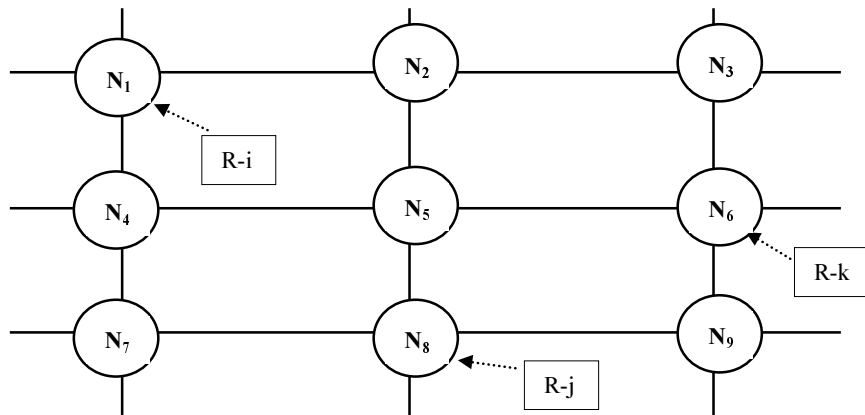


Figure 3. Communication Network Access Points

The appearance of tokens in places P3, P4, P7, P2, and P5 in Figure 2 represents the information flow between tasks: ‘information (i,j) ’, ‘information (i,k) ’, ‘control (j,k) ’, ‘control (j,i) ’, and ‘information (k,j) ’ in Figure 1. In turn, traffic messages with source-destination node pairs (N_1, N_8) , (N_1, N_6) , (N_8, N_6) , (N_8, N_1) , and (N_6, N_8) in Figure 3 represent the physical message passing, respectively.

2.3 Step 3: Create Message Dependency Relations

When the firing sequence of transitions is known, we can determine the sequence of message passing. When the firing sequence is not known, we have to consider all possible sequences of transitions. These all possible sequences are captured in a full occurrence graph of a Petri net. Generating a full occurrence graph and searching it exhaustively are not practically acceptable when the state space is large.

In a conflict free net, firing of one transition does not disable other transitions. Once tokens in the preset of a transition fulfill the enabling conditions, the transition will definitely fire under a “must” firing rule. So, even though multiple transitions in a Petri net are enabled concurrently, the final markings of the net are the same regardless of the firing order of the concurrently enabled transitions. Also, the group of messages that are produced after firing of concurrently enabled transitions have no precedence relations inside the group of messages. Message dependency relations occur only between different sets of messages that are produced after subsequent firings other than concurrent firings. Thus, given a conflict free net with initial markings, one sequence of occurrence of markings in a partial occurrence graph has all the information necessary to count the number of messages in the system and their precedence relations. If we generate a partial occurrence graph in the depth-first search algorithm and there is no circuit in the net, it is not costly to capture the message dependency relations, and the methodology developed can be used

effectively.

If a Petri net is not conflict free, however, the net may have multiple final markings. Even if the net has a single final marking, there may exist multiple sequences of occurrences of markings that have different precedence relations amongst different sets of messages. To handle those non-conflict free nets, the methodology may require repeated simulations of the communications network corresponding to the different paths in the full occurrence graph. One alternative way to estimate network delays between all transitions is to set the initial marking so that all transitions are covered in each path of the full occurrence graph. Then, the methodology can estimate the network delay between tasks by selecting one path. Assume that the communications network has the following characteristics: (1) network delays are longer at its heavier traffic loads than its lighter traffic loads, and (2) the degree of traffic loads is defined as the amount (e.g., number or size) of messages in a given time interval. By selecting the path that has the maximum number of messages (most likely the longest path), the methodology can estimate the upper bound of network delays given the initial marking. Similarly, by selecting the path that has the minimum number of messages (most likely the shortest path), the methodology can estimate the lower bound of network delays given the initial marking. Instead of repeated simulations of a communications network as many as the number of paths, the alternative method may be efficiently used for a worst-case timing analysis.

The newly generated tokens are identified by subtraction of multi-sets between subsequent markings. However, untimed occurrence graphs can not distinguish the new tokens in a self-loop. In a model of a real-time system, new tokens in a self-loop can be distinguished by their time stamps. So we can have a tool to distinguish new tokens in a self-loop by adding a counter or a logical time [10] unit. In our methodology, we used a logical time unit because it is easy to analyze the logical behavior of a system. Using the Occurrence Graph Analyzer (OGA) embedded in Design/CPN, a timed Occurrence Graph can be converted into an untimed Occurrence Graph easily. By adding time delay “1” on the arc from transition “T1” to place “P11”, if necessary, we can distinguish the new tokens in the self-loop. If we generate an occurrence graph after assigning time delays to transitions, the occurrence graph becomes a timed occurrence graph. But the order of message passing does not change in a conflict free net. Also a timed occurrence graph is a subset of an untimed occurrence graph [11]. So in a conflict free net, we can extract the number of messages and their precedence relations without affecting the logical behavior. Now we are ready to extract the message precedence relations by a simple set operation; subtraction of multi-sets.

Suppose that an Ordinary Petri net is represented by a quadruple (P, T, I, O) where:

P = {p₁, p₂, ..., p_n} is a finite set of places.

T = {t₁, t₂, ..., t_m} is a finite set of transitions.

I is an input mapping P x T → {0, 1} corresponding to the set of directed arcs from P to T.

O is an output mapping T x P → {0, 1} corresponding to the set of directed arcs from T to P.

Given the current marking M, firing of the transition t results in a new marking M' where:

$$M'(p_i) = M(p_i) - I(p_i, t) + O(t, p_i), \text{ For } i = 1, \dots, n.$$

So, if we perform an operation of subtraction; M'(p_i) - M(p_i), the computation results in {1, 0, -1} from

$O(t, p_i)$	-	$I(p_i, t)$	=	$M'(p_i) - M(p_i)$
0		0		0
0		1		-1
1		0		+1
1		1		0

The “+” and “-“ signs tell whether the token is a newly placed one or a removed one. The “0” at the fourth row occurs when the structure has a self-loop. It must be detected because one token is removed and one token is newly generated. But it is not distinguished from the “0” at the first row. That represents no change after firing of the transition. By adding time stamps, the “0” at the fourth row can be distinguished from the “0” at the first row.

Let the service times of physical resources R-i, R-j, and R-k (i.e., an execution time of a task executor) be τ_{11} , τ_{12} , τ_{13} that are associated with each place P11, P12, and P13, respectively. Also let the network delays of messages $m(p3)$, $m(p4)$, $m(p7)$, $m(p2)$, and $m(p5)$ be δ_3 , δ_4 , δ_7 , δ_2 , and δ_5 associated with each place P3, P4, P7, P2, and P5, respectively. A message dependency graph corresponding to Figure 2 can be depicted as shown in Figure 4.

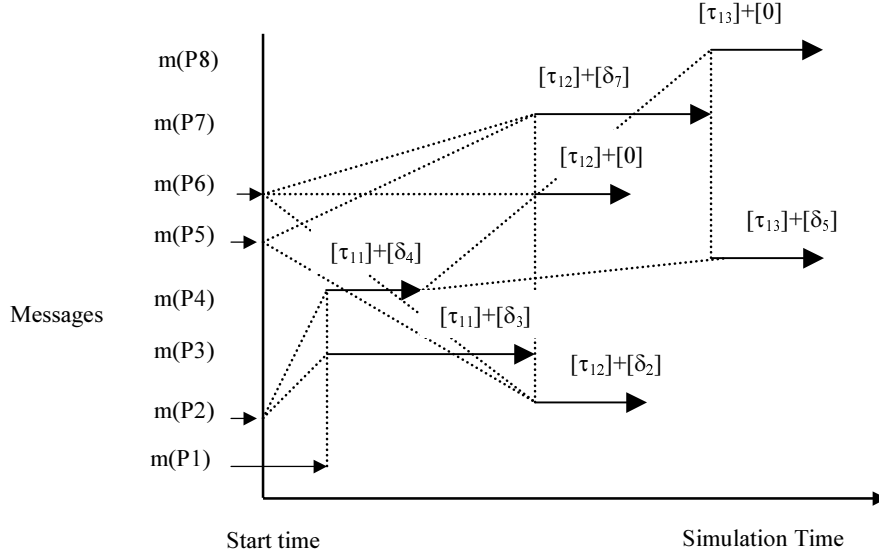


Figure 4. An Example of Message Dependency Graph

2.4 Step 4: Simulate Communications Network

The role of message dependency (or precedence) relations is to determine the order of message passing to schedule every message over a single time line during the simulation. If we know the network delay, the messages can be linearly ordered over a single time line: total order. Since the delays are not known at the logical model, the messages from the logical model only show a partial order. Network delays are estimated during simulation and they are used for scheduling the next messages.

The figure shows that two messages $m(p3)$ and $m(p4)$ (i.e., tokens produced at places P3 and P4, respectively, after firing of transition T1) will be triggered by two initial messages $m(p1)$ and $m(p2)$. In the figure, the initial time stamp of message $m(p1)$ is bigger than the initial time stamp of message $m(p2)$. If transition T1 takes an action delay (e.g., τ_{11}), messages $m(p3)$ and $m(p4)$ must be generated after the action delay. In turn, each message $m(p3)$ and $m(p4)$ will be used for triggering the next messages. The second message of place P2 $m(p2)$ will be generated after the arrival of $m(p3)$ when the time values of initial messages $m(p5)$ and $m(p6)$ are passed. Place P6 in Figure 2 is self-looped. So message $m(p6)$ does not involve network delay (0 delay). Message $m(p7)$ will be generated after arrivals of $m(p3)$, $m(p5)$, and $m(p6)$ at the latest arrival time of all 3 messages. Finally, messages $m(p5)$ and $m(p8)$ will be triggered by two messages $m(p7)$ and $m(p4)$. In this specific example, it shows a case that message $m(p7)$ generated at resource R-j (i.e., source node N8) arrives at resource R-k (i.e., destination node N6) later than message $m(p4)$ generated at resource R-i (i.e., source node N1) arrives at resource R-k (i.e., destination node N6). But $m(p8)$ has no network delay (0 delay) because it is a sink of information in the model.

Table 1 is an example of a traffic log file. Instead of having a message generation time, which is usually specified by a probabilistic form such as Poisson or Exponential in a typical communication network simulation approach, each message is specified by its dependency relations.

Table 1. Message Traffic Log

ID	source	destination	size	priority	protocol	dependency
1: $m(p_1)$	INITIAL	N1	1000byte	0	UDP	[]
2: $m(p_2)$	INITIAL	N1	1000byte	0	UDP	[]
3: $m(p_5)$	INITIAL	N8	1000byte	0	UDP	[]
4: $m(p_6)$	INITIAL	N8	1000byte	0	UDP	[]
5: $m(p_3)$	N1	N8	1000byte	0	TCP	[1,2]
6: $m(p_4)$	N1	N6	1000byte	0	TCP	[1,2]
7: $m(p_2)$	N8	N6	1000byte	0	TCP	[3,4,5]
8: $m(p_6)$	N8	N8	1000byte	0	TCP	[3,4,5]
9: $m(p_7)$	N8	N6	1000byte	0	TCP	[3,4,5]
10: $m(p_5)$	N6	N8	1000byte	0	TCP	[6,9]
11: $m(p_8)$	N6	EXTERNAL	1000byte	0	UDP	[6,9]

2.5 Step 5: Estimate Network Delays

After running the simulation, two types of delay information can be obtained. One is the statistical data, and the other is a pair of bounded network delays, if it exists. Assume that the communications network has the characteristics explained in section 2.3. If the delays with light traffic loads (lower bound network delays) are smaller than the delays with heavy traffic loads (upper bound network delays), and if they are consistent for various degrees of traffic loads, the two assumptions regarding the characteristics of the communications network may be considered reasonable. Then, we can determine the minimum and maximum network delay pairs for each message and use them to verify the end-to-end timing constraints.

If the simulation results show a random pattern, we can not determine the bounds. Instead, the network delay pattern will be formalized in a statistical form such as average delay time and variance. This probability distribution does not express the lower or upper bound delays. Consequently it is not decidable whether the end-to-end timing constraint of a system is met or not. The timing behavior of the system will be predicted by only repeated simulations of the Petri net model or the communications network model.

2.6 Step 6: Introduce Time to Executable Functional Architecture

Once we obtain the network delay pattern with either a probabilistic distribution or bounded delays, these network delay values and service times are specified as arc expressions of a Petri net. The resulting Timed Petri net model has all information to predict the performance of the system. This is the last step to develop the performance prediction model. If we care only about the end-to-end timing delay, this step is not necessary since the output data from communication model has the same information. However, this step helps to analyze the logical behavior and timing behavior together using the resulting timed occurrence graph.

3. DESIGN OF PERFORMANCE PREDICTION MODEL GENERATOR

The Performance Prediction Model Generator (P2MG) was designed based on the requirements so that it can be used efficiently and effectively for performance prediction of a real-time system for various scenarios or design options. Figure 5 shows the workflow model. In this methodology, the functional models are assumed to exist.

The scenario is expressed as initial conditions of the functional model (i.e., initial markings of the Petri net). The state of a functional model is represented by an occurrence graph of the Petri net. Vector $[m(p_1), m(p_2), m(p_3), \dots]^T$ in the functional model represents the state of a system in the functional view. Using the OGA of Design/CPN, the logical behavior of the system is verified first. Then, from the

occurrence graph, the order of message passing is captured and generated in a traffic log file. The order of message passing $[\{a,b\} \rightarrow \{c,d\} \rightarrow \{e\}, \dots]$ has information about the number of messages and the precedence relations between sets of messages. The communication network simulator processes each message in a sequential order over a single time line, and generates all the delay values for each message. The result shows node processing delays (i.e., pre- and post- processing delays) and network transmission delays for all messages. Finally, all the delay values are mapped back to the functional model for analysis of both logical and timing properties using a timed occurrence graph.

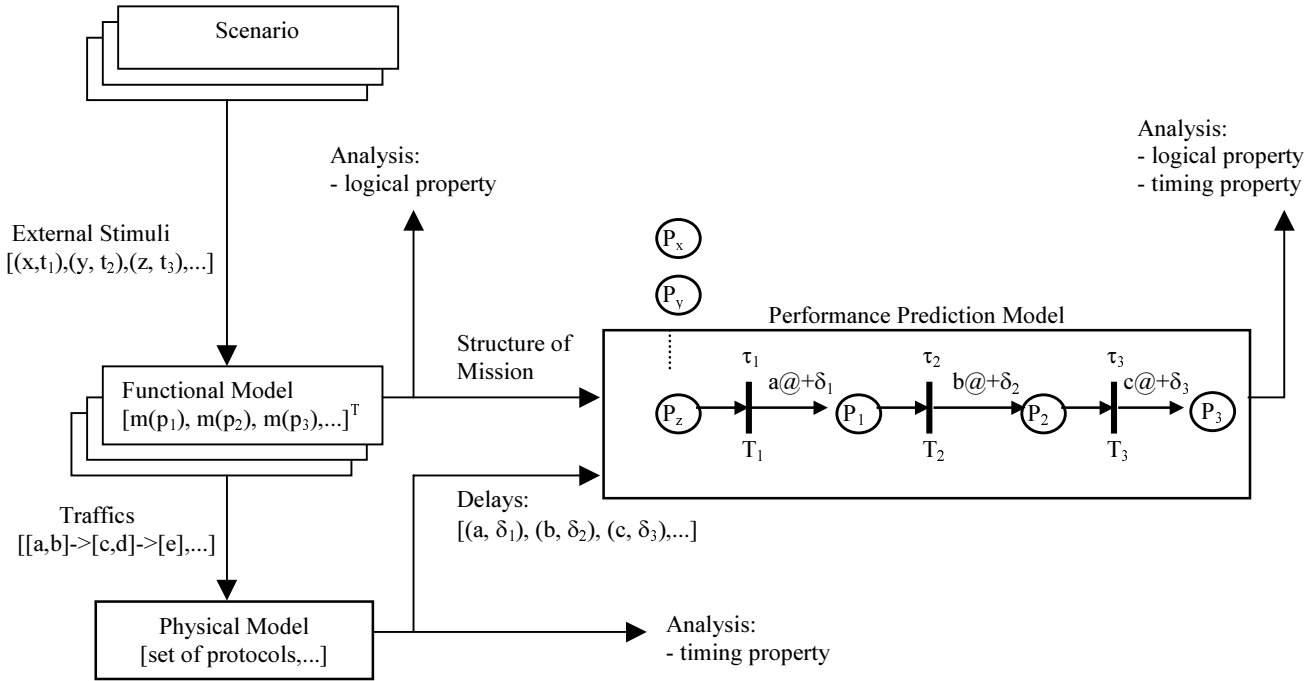


Figure 5. Workflow Model for P2MG

To implement the methodology, we used Design/CPN for the description of the functional model and NS for the description of the physical model. The P2MG consists of 5 major software components; Reachability Tree Generator (RTG), Message Generator (MG), Performance Prediction Model Converter (PPMC), Network Topology Generator (NTG), and Communication Network Simulation Script (CNSS). The four components RTG, MG, PPMC and NTG were implemented using the ML (Meta Language) imbedded in Design/CPN. CNSS was implemented using the TCL (Tool Command Language) embedded in NS.

To describe each component, we will use a simple Petri net as shown in Figure 6 (with the global declaration of Table 2). The Petri net corresponds to Figure 1, but very simplified for ease of understanding. The structure of the model describes a typical command and control process. Let us develop an arbitrary scenario. The system senses threats using radar. It scans the battlefield periodically, 30 times per minute (e.g., the minimum action delay of sensing is 0 seconds and the maximum action delay is 2 seconds). It disseminates the tactical picture to its missile Fire Direction Center (FDC). Once the FDC receives information of a new threat, the FDC checks the availability of weapons, and issues an order to a Missile Launcher following the engagement rule, and issues the surveillance directive. This action is invoked automatically by embedded algorithms in the automated missile controller. It takes some time for data fusion, but the delay is assumed to depend only on the size of the tactical picture received and the bit processing power of the computing device. The missile launcher platform requires 10 seconds warming time to fire a missile (e.g., minimum delay of 0 seconds when it is already warm and maximum action delay of 10 seconds). All messages in the system are 1000 bytes (8000bits) long.

3.1 Reachability Tree Generator (RTG)

The RTG primarily uses software function codes provided by the OGA of Design/CPN. However, this component was implemented to be interactive with users. It generates a reachability tree, searches final states, and calculates the shortest and longest paths from the initial state to final states. This occurrence graph (OG) generation module generates a partial reachability tree until it finds at least one final state. If the net is conflict free, this partial reachability tree can be used to capture all the necessary state information. The current version of the OGA in Design/CPN uses “breadth-first” search algorithm to build the OG. But it provides “narrow” search; a kind of “depth-first” search. Using this option can reduce computation time to find a final state. If the net is not conflict free, the user will be provided an option to search all final states and select one of the paths from the initial state to multiple states.

3.2 Message Generator (MG)

The main feature of MG is to generate the order of message passing containing their precedence relations. It generates a traffic log file in the format shown in Table 1 so that it can be used by the CNSS. MG provides the user with a Graphic User Interface (GUI) developed in the Design/CPN environment in order to obtain information necessary to make the traffic log file.

The MG searches what resources are allocated to which transitions in order to generate its source and destination nodes of the communication network. The mapping mechanism is based on the dual roles of resources as described in Section 2 (places P11, P12, and P13 in Figure 2 are placeholders to map functional architecture and physical architecture). The process of selecting alternative resources is an issue of resource allocation or an issue of design. So we can compare the alternatives in order to execute the same scenario. This GUI helps to select instances of resources that are connected to the communication network as leaf nodes.

At this module, the user enters the attributes of actors, which are the performers of tasks. The main attribute is the action time. This action time is differentiated from the node processing delay and network transmission delays. Basically, those node processing and network delays are associated with the capability of resources whose performance parameters are specified in the physical model. The delay of an actor is associated with the delay of task executor(s). If every task is an autonomous data processing task, the actor’s delay may be set to “0”. Once this attribute is obtained, it is maintained at a special region of each transition and arc. It can be alternatively selected depending on scenarios. Another attribute is the demand protocol to transmit each message.

3.3 Performance Prediction Model Converter (PPMC)

The main purpose of the Performance Prediction Model Converter is to transform the logical Petri net into a Timed Petri net (i.e., Performance Prediction Model). An example of the transformed Timed Petri net model is shown in Figure 7. Compare the two logical and Timed Petri nets; Figure 6 is the logical Petri net and Figure 7 is the transformed Timed Petri net. Figure 7 additionally has a local declaration node, a time delay arc expression, a code segment, and a temporary declaration node. They are created automatically by PPMC.

The local declaration node defines the variables of delay values. Each variable name tells a logical connection link name between two tasks. It consists of a transition name (source) and a place name (sink). For example, a variable name AAW’T1_1’AAW’P3 means that the first instance of transition T1 at the AAW page sends messages to place P3 at the AAW page. Those assigned integer values to each variable are message identification numbers. So the total number of messages in a link represents the total number of newly created messages. For example, the transition T1 generated 3 tokens (i.e., 9,10,11) and places P3, P4, and P11 received 1 token each from transition T1.

The existence or non-existence of code segments tells whether that transition fired at least one time or never fired in the scenario (i.e., initial marking). The non-existence of the code segment at each transition means that the transition has not fired at all given the initial marking, or the transition has self-

loop. This helps visualizing the logical behavior of the model.

The delay variables are automatically added at the tail of each arc expression with time expression. The "fnDELTA" on arc expression is the delay selection function. If a time delay expression appears on an arc, the arc's ending place must be declared as a timed color set.

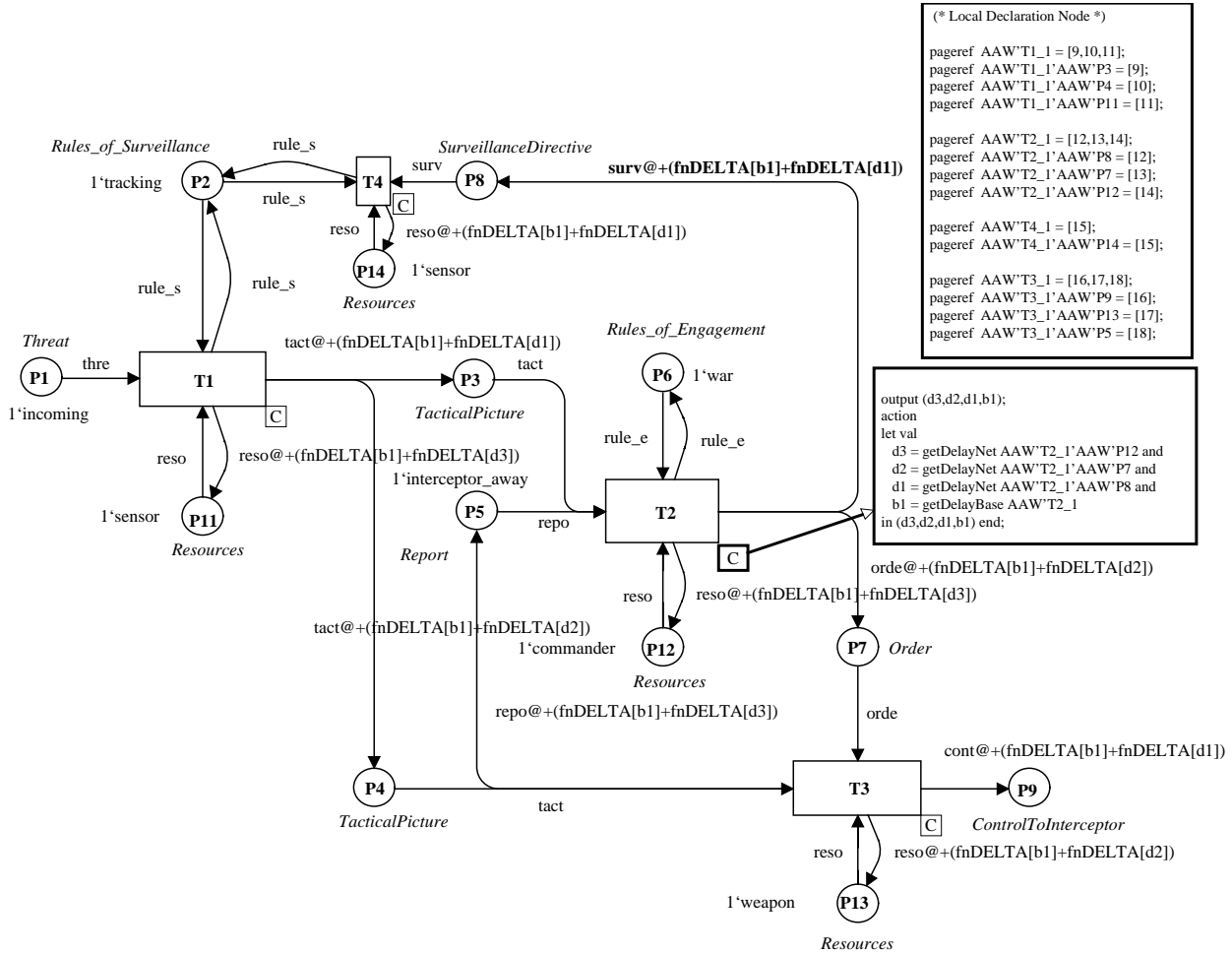


Figure 7. Intermediate Performance Prediction Model

The delay variables of the local declaration node and the delay variables of arcs are mapped in the code segment. In the code segment, another two delay retrieving functions are used with names "getDelayBase" and "getDelayNet". These are used to retrieve the delay values from the local declaration node and send them to each arc. The functions associated with delays (i.e., getDelayBase, getDelayNet, and fnDELTA) are defined at the temporary declaration node (Table 3). These functions can be also defined as user's preference. By default, they were defined to get the minimum delay, maximum delay, zero delay, and random delays. The user can select this option anytime during simulation of the Petri net.

Once the net is converted into a performance prediction model, the net can be executed "as is". In this case, the net will consider the message ID as its own delay value. By doing this, the user can check if the system is modeled correctly before stepping forward. One example of this is as follows. Assume that the scenario changes so that the FDC changes the surveillance directive from the scanning mode to the tracking mode once the object is identified as a hostile flight. Transition T4 updates the new directive from the old surveillance directive to the new surveillance directive by changing the arc inscription. The

resulting performance prediction model will create a time expression on the arc from T4 to P2. If we have the place P2 untimed, it will raise a syntax error. Even if the place represents a rule, the rule changes over time, involves updating time of the rule, and hence the place must be defined as a timed color set.

Table 3. Temporary Declaration Node

```
(* Temporary Declaration Node *)

globref MinOrMaxBase = "Max" ;
globref MinOrMaxNet = "Max" ;

color timescale = int;
val unittime = (0: timescale) ;
val minmaxdef = (~1: timescale) ;
var d3,d2,d1,b1 : timescale ;

fun Zero (xs: timescale list) = unittime;
fun sum (nil) = unittime
  | sum ((x:timescale)::xs) = x+sum(xs);
local
  fun min value nil = if value = minmaxdef then unittime else value
    | min value ((x:timescale)::xs) =
      let val new_value = if value > x orelse value = minmaxdef then x else value
        in min new_value xs end;
in fun Min (xs: timescale list) = min minmaxdef xs
end;
local
  fun max value nil = if value = minmaxdef then unittime else value
    | max value ((x:timescale)::xs) =
      let val new_value = if value < x orelse value = minmaxdef then x else value
        in max new_value xs end;
in fun Max (xs: timescale list) = max minmaxdef xs
end;

exception TimeNotDetermined ;
fun getDelayBase (xs: timescale list ref) = if !xs=[] then raise TimeNotDetermined
  else if (!MinOrMaxBase) = "Zero" then Zero (!xs)
  else if (!MinOrMaxBase) = "Min" then Min (!xs)
  else if (!MinOrMaxBase) = "Max" then Max (!xs)
  else random (list_to_ms (!xs));

fun getDelayNet (xs: timescale list ref) = if !xs=[] then raise TimeNotDetermined
  else if (!MinOrMaxNet) = "Zero" then Zero (!xs)
  else if (!MinOrMaxNet) = "Min" then Min (!xs)
  else if (!MinOrMaxNet) = "Max" then Max (!xs)
  else random (list_to_ms (!xs));

fun fnDELTA (xs: timescale list) = nth (xs,0);
```

Once the delay values are obtained from the physical model, the message IDs are replaced with their delay values. Figure 8 shows the final performance prediction model that was derived from the logical Petri net model (i.e., Figure 6) using the output delay data (shown in Table 4).

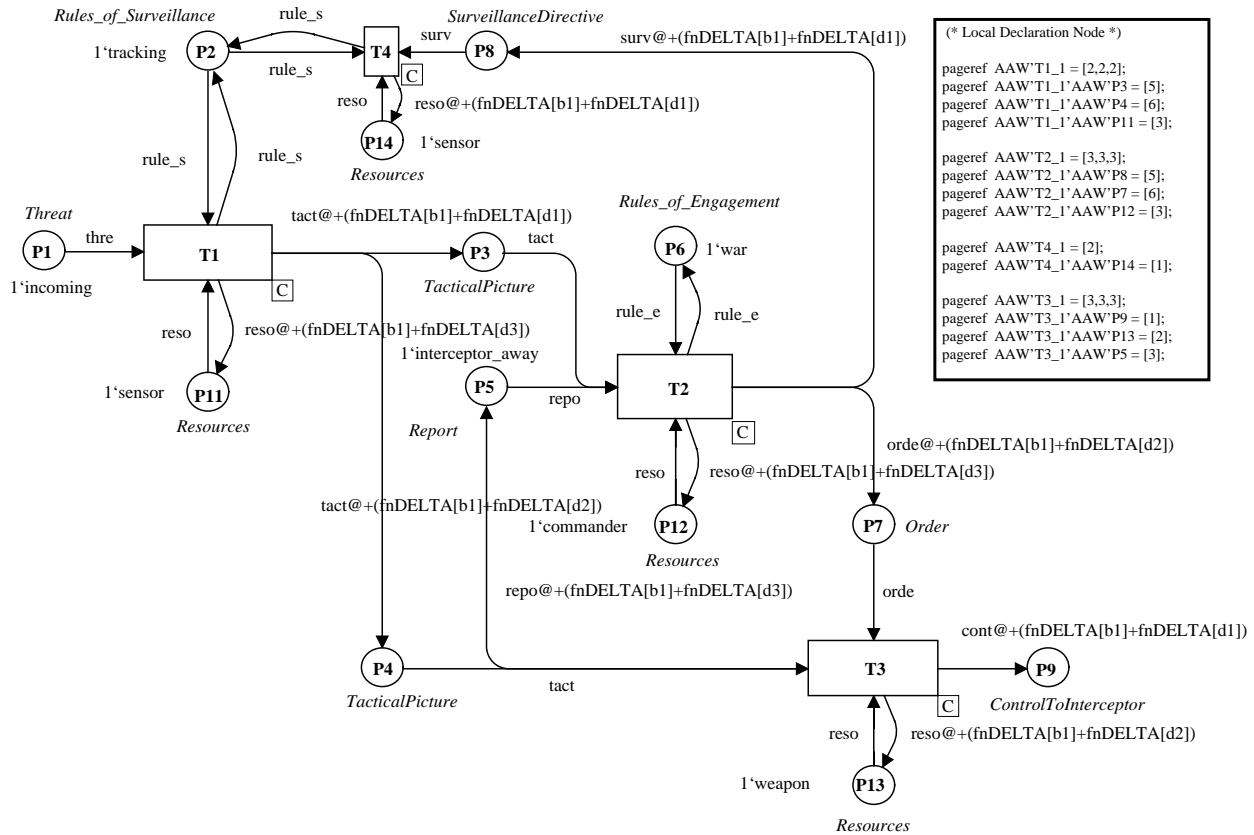


Figure 8. Final Performance Prediction Model

Table 4. Output Delay Data from Communication Model produced by CNSS

```

model name: /demo/
Original number of message: /18/
Valid number of message: /16/
variables: /("msg_id", ["pre_proc_delay", "action_time", "post_proc_delay", "net_delay"])/

(1,[0.000000,0.000000,0.000000,0.000000])
(3,[0.000000,0.000000,0.000000,0.000000])
(5,[0.000000,0.000000,0.000000,0.000000])
(6,[0.000000,0.000000,0.000000,0.000000])
(7,[0.000000,0.000000,0.000000,0.000000])
(8,[0.000000,0.000000,0.000000,0.000000])
(9,[2.000000,0.000000,1.000000,4.000000])
(10,[2.000000,0.000000,2.000000,4.000000])
(11,[2.000000,0.000000,3.000000,0.000000])
(12,[3.000000,0.000000,1.000000,4.000000])
(13,[3.000000,0.000000,2.000000,4.000000])
(14,[3.000000,0.000000,3.000000,0.000000])
(15,[2.000000,0.000000,1.000000,0.000000])
(16,[3.000000,0.000000,1.000000,0.000000])
(17,[3.000000,0.000000,2.000000,0.000000])
(18,[3.000000,0.000000,3.000000,0.000000])

```

The delay values were imported into the Timed Petri net model as a list of values. CNSS can simulate the physical model repeatedly with the same order of message passing with various options and

can generate multiple instances of delay values. However, the resulting delay values are imported into the functional model without processing them in a statistical form. The main purpose of using raw delay data is to generate the timed occurrence graph effectively. Another reason is to allow the user to be able to select those values as his analysis purpose. The converted performance prediction model has options how to use those values as specified in the temporary declaration node.

3.4 Network Topology Generator (NTG)

The purpose of the NTG is to generate network topology since the current NS does not provide a GUI for topology generation. The main use of this tool is, however, to allow the user to select the physical resources interactively depending on the design option or resource allocation option for each scenario. NTG provides the user with a GUI to enter the number of nodes and their attributes; the number of switching nodes of the Wide Area Network (WAN), the number of terminal nodes of each Local Area Network (LAN), and the number of leaf nodes at which the physical resources are connected to the WAN node. It automatically lays out the nodes in a two-dimensional grid and connects them in a default network topology of tactical communication networks. Figure 9 is an example of a network topology drawn by NTG.

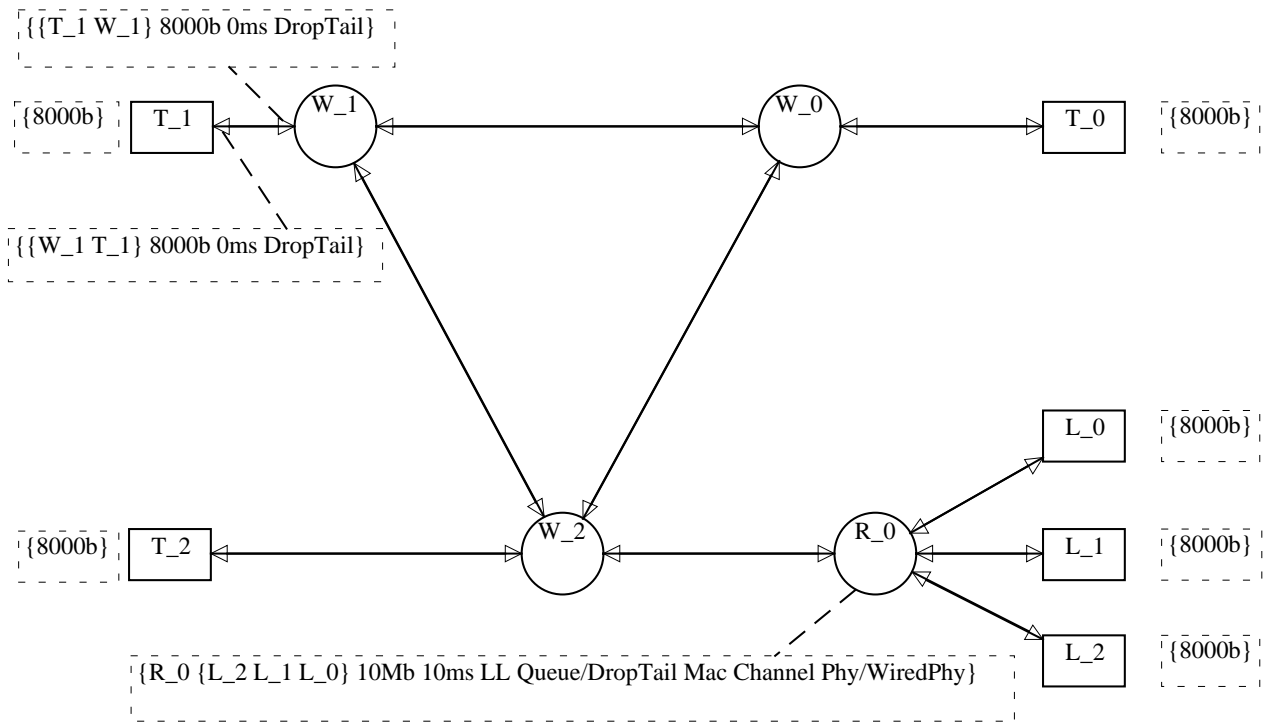


Figure 9. An Example of Network Topology drawn by NTG

Figure 9 shows three WAN nodes (W_0 , W_1 , and W_2), one LAN router (R_0) with three terminals (L_0 , L_1 , and L_2), and another three terminal leaf nodes (T_0 , T_1 , and T_2). All links except LAN have link attributes with data transmission rate of “8000bps” and “0ms” electronic propagation delay. The example specification of the LAN is a wired, 10Mbps Ethernet protocol. Since the size of every message in the system was specified as 1000 bytes (8000bits) long in section 3, every delay of one hop transmission between two nodes will have 1 second delay. Also the figure shows that every leaf node (T_i or L_i) has bit processing power of “8000bps”. So it will take another 1 second for data processing at each node.

NTG defines and uses special node and region types that are different from “PetriNode” types.

Once nodes or links are constructed, the user can modify the topology of the network and its attributes. The user can add and delete a link using the drawing function (“connector”) of Design/CPN. Once the topology is constructed, the tool performs syntax check and produces a network topology file in a format so that the CNSS can read it directly.

3.5 Communication Network Simulation Script (CNSS).

NS provides an object-oriented, event-driven simulation engine with various protocol modules. CNSS basically uses those facilities except for the node processing module. The developed methodology and tools support description of 3 types of delay: Node processing delay (pre- and post- processing delays) of the computing device, action delay of the task executor, and transmission delay of the network. Node processing time describes two types of data processing delays: pre-processing delay and post-processing delay. Pre-processing delay is a delay between the time at which all conditioned information arrives and the time at which the decision making task starts. Assume that sensor *A* reports a large image data for the feature of a detected object and sensor *B* reports a small signal data for the location of the object. The time to display the information to decision maker will vary. Post-processing delay is the delay between the time at which decision is made and the time at which the order is to start broadcasting. To handle those pre- and post- processing delays, CNSS creates another terminal node and attaches to the corresponding network node. The bit processing power of the computing devices is specified as another bandwidth of a special link between those two nodes. The node processing delay is now measured by the link delay. Assume that a transition supported by the corresponding computing device has 3 input places and 1 token at each place and 3 tokens are 1000, 2000, and 3000 bytes long each. If the bit processing power of the computing device is 1000bps, the pre-processing time will 48 seconds; $6000 \text{ (byte)} * 8 \text{ (bit/byte)} / 1000 \text{ (bit /second)} = 48 \text{ seconds}$. If the same transition has 2 output places, produces 2 tokens at each place, and the token at the first place is 100 bytes long and that at the second place is 1000 bytes long, then the post-processing delay will be 17.6 seconds; $\{2 \text{ (token)} * 100 \text{ (byte/token)} * 8 \text{ (bit/byte)} + 2 \text{ (token)} * 1000 \text{ (byte/token)} * 8 \text{ (bit/byte)}\} / 1000 \text{ (bit/second)} = 17.6 \text{ seconds}$. By expressing distance and speed of a vehicle as *meter* and *meter per second* respectively, it can model a force maneuvering in a war-game model. Similarly, by expressing the data as workload, it can model a manufacturing system, business processing system, or logistics system.

NS produces a large amount of simulation data because it executes protocols at each layer in detail. For example, the resulting simulation data of the detailed model of Figure 8 was about 250Mbs long. CNSS creates a monitoring agent class and measures the network delays during simulation, and produces only that delay information. These monitored delays are also used for scheduling the next messages in compliance with message dependency relations produced by MG of section 3.2 in the format of Table 1.

4. RESULTS

Now let us specify all action time delays of task executors to be “0” delay. That is to select the minimum delay from {0s, 2s} for sensor execution time and the minimum delay from {0s,10s} for Missile’s warming time. Resources “sensor”, “commander” and “weapon” in Figure 6 are mapped to terminals T_0, T_1, and T_2 in Figure 9 respectively. Under those specifications and scenario in the previous section, Table 5 was produced from the Performance Prediction Model (Petri net). Table 6 was produced from the Physical Model (NS). The time stamps of messages in Table 5 and the arrival times of messages in Table 6 show how the system responded to external stimuli. For an incoming threat (message id 1) at time 0 second, a weapon fired at time 20 second (message id 16), the missile launcher repositioned at time 21 second (message id 17), and the after-action-report arrived at time 22 second (message id 18) at the FDC. Table 5 and Table 6 have the same information regarding the timing behavior.

Table 5. Message List produced from Figure 8

```

*** Numbered Message List of model : /home/ishin/funcbin/data/././workshop/demoafter***
Msg_ID  Content      Time_Stamp  From_Transition  To_Place
1      incoming      0           INITIAL          AAW'P1 1
2      war           0           INITIAL          AAW'P6 1
3      weapon        0           INITIAL          AAW'P13 1
4      tracking       0           INITIAL          AAW'P2 1
5      sensor        0           INITIAL          AAW'P11 1
6      interceptor_away 0           INITIAL          AAW'P5 1
7      commander     0           INITIAL          AAW'P12 1
8      sensor        0           INITIAL          AAW'P14 1
9      new           7           AAW'T1 1        AAW'P3 1
10     new           8           AAW'T1 1        AAW'P4 1
11     sensor        5           AAW'T1 1        AAW'P11 1
12     default       15          AAW'T2 1        AAW'P8 1
13     intercept     16          AAW'T2 1        AAW'P7 1
14     commander     13          AAW'T2 1        AAW'P12 1
15     sensor        18          AAW'T4 1        AAW'P14 1
16     take_off      20          AAW'T3 1        AAW'P9 1
17     weapon        21          AAW'T3 1        AAW'P13 1
18     interceptor_away 22          AAW'T3 1        AAW'P5 1

```

Table 6. Delay Output from Ns

```

modelname: demo
Original number of message: 18
Valid number of message: 16
msg-id platform start-pre  start-p  send-m  rcv-m  pre-proc-dly  action-t  post-proc-dly  net-dly
1  INITIAL  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
3  INITIAL  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
5  INITIAL  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
6  INITIAL  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
7  INITIAL  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
8  INITIAL  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
9  T_0      0.000000  2.000000  3.000000  7.000000  2.000000  0.000000  1.000000  4.000000
10 T_0      0.000000  2.000000  4.000000  8.000000  2.000000  0.000000  2.000000  4.000000
11 T_0      0.000000  2.000000  5.000000  5.000000  2.000000  0.000000  3.000000  0.000000
12 T_1      7.000000  10.000000  11.000000  15.000000  3.000000  0.000000  1.000000  4.000000
13 T_1      7.000000  10.000000  12.000000  16.000000  3.000000  0.000000  2.000000  4.000000
14 T_1      7.000000  10.000000  13.000000  13.000000  3.000000  0.000000  3.000000  0.000000
15 T_0      15.000000  17.000000  18.000000  18.000000  2.000000  0.000000  1.000000  0.000000
16 T_2      16.000000  19.000000  20.000000  20.000000  3.000000  0.000000  1.000000  0.000000
17 T_2      16.000000  19.000000  21.000000  21.000000  3.000000  0.000000  2.000000  0.000000
18 T_2      16.000000  19.000000  22.000000  22.000000  3.000000  0.000000  3.000000  0.000000

```

Figure 10a is the full occurrence graph of the functional architecture (logical Petri net, Figure 6). Figure 10b is the full occurrence graph of the performance prediction model (Timed Petri net, Figure 8). The reachable state of the system has been reduced from two paths to one path. Notice that Figure 10b shows the same value of time stamps as shown in Table 6.

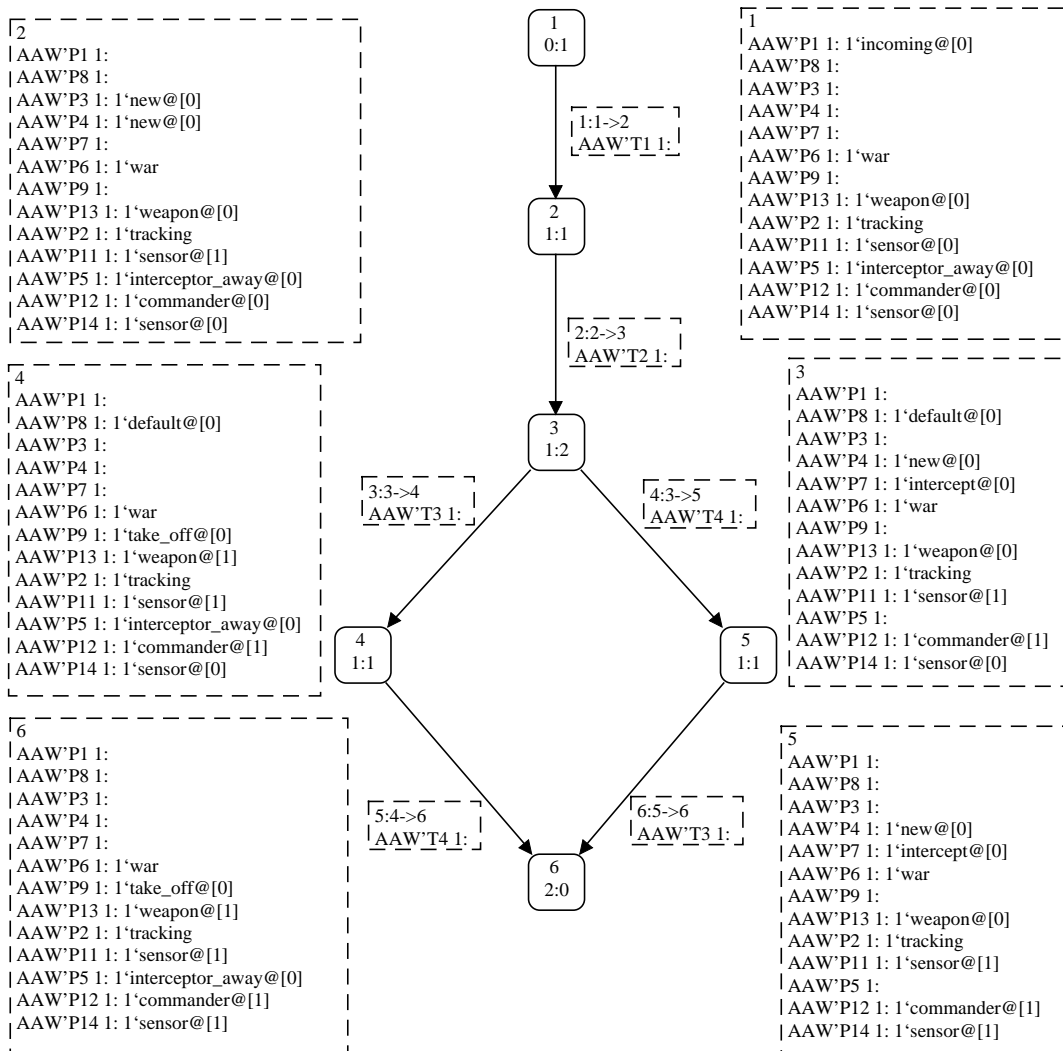


Figure 10a. A Full Occurrence Graph of Logical Model (Petri net in Figure 6)

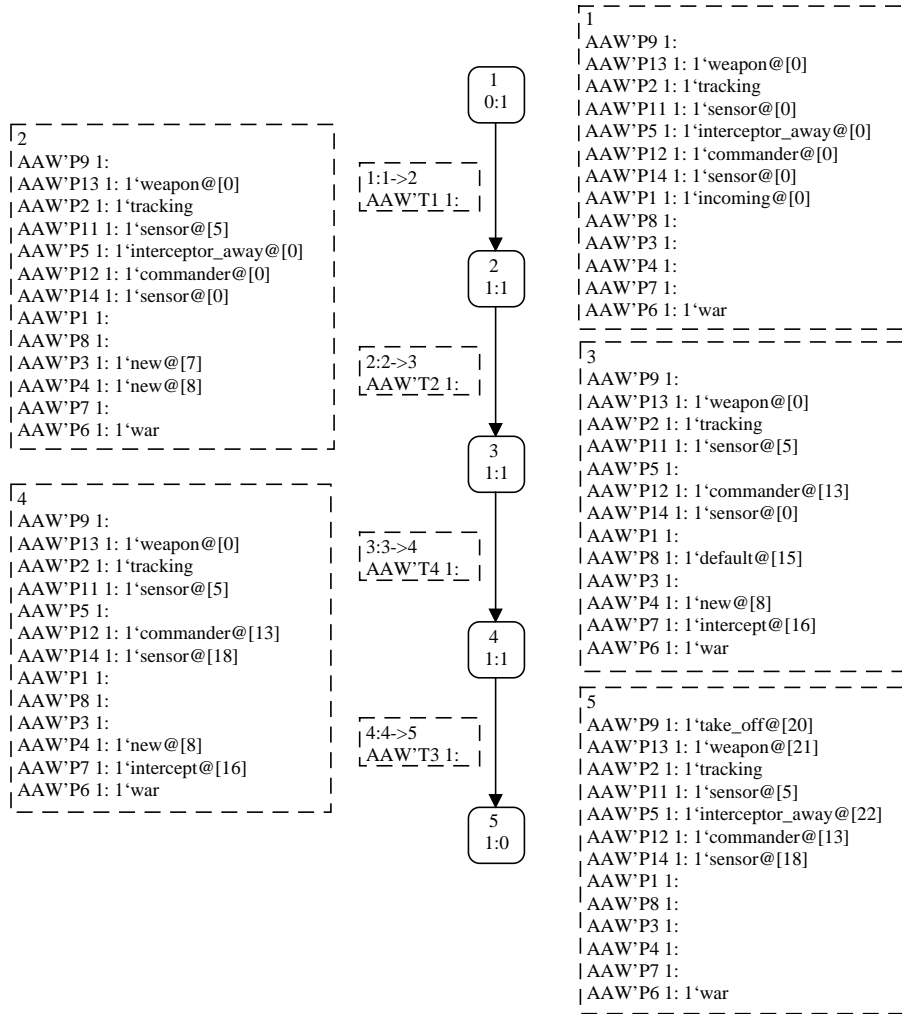


Figure 10b. A Full Occurrence Graph of Performance Prediction Model (Petri net in Figure 8)

5. CONCLUSION

The methodology developed provides an effective performance evaluation method for both qualitative (logical property) analysis and quantitative (timing property) analysis. The P2MG was implemented in the Design/CPN environment for automatic generation of performance prediction model. It is powered by the OGA of Design/CPN. The application of this methodology and the software tool needs only one simulation of each model, if the Petri net is conflict free. However, it is not limited to conflict free nets. It can be used for other Petri nets by repeated simulations. The current version of the software tool is limited to a system that uses a common physical architecture with multiple functional architectures or missions. The next step is to expand the tool to apply to heterogeneous physical architectures.

ACKNOWLEDGMENT

We appreciate the valuable comments of the anonymous reviewers.

REFERENCES

- [1] Design/CPN, <http://www.daimi.au.dk/designCPN/>
- [2] Network Simulator - ns (version 2), <http://www-mash.cs.berkeley.edu/ns/ns.html>
- [3] Tsai, Jeffrey J.P. and S.J. Yang (1995), *Monitoring and Debugging of Distributed Real Time Systems*, Washington, D.C., IEEE Computer Society Press.
- [4] Stankovic, J.A., K. Ramamritham and S. Cheng (1985), "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," *IEEE Transactions on Computer*, Vol. 34, Dec., pp. 1130-1143.
- [5] Gerber, Richard, Seongsoo Hong and Manas Saksena (1994). "Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes," *Proceedings of Real-Time Systems Symposium*, 7-9 Dec., pp. 192-203.
- [6] Deng, Yi, Jiacun Wang and R. Sinha (1998). "Incremental Architectural Modeling and Verification of Real-Time Concurrent Systems," *Proceedings of Second International Conference on Formal Engineering Methods*, 9-11 Dec., pp. 26-34.
- [7] Felder, M., D. Mandrioli and A. Morzenti (1994). "Proving Properties of Real-Time Systems Through Logical Specifications and Petri Net Models," *IEEE Transactions on Software Engineering*, Vol. 20, No. 2, Feb., pp. 127-141.
- [8] Levis, Alexander H. (1992). "A colored Petri Net model of intelligent nodes", *Robotics and Flexible Manufacturing Systems*, J.C. Gentina and S.G. Tzafestas (Editors), Elsevier Science Publishers B.V. (North-Holland), pp. 369-379.
- [9] Fahmy, Hany I. and C. Douligeris (1996). "NAMS: Network Automated Modeler and Simulator," *Proceedings of the 29th Annual Simulation Symposium*, SIMULATION '96, April, pp. 65-70.
- [10] Lamport, L (1978). "Time, Clocks and Ordering of Events in a Distributed System," *Communications of the ACM*, Vol. 21, No. 7, pp. 558-565.
- [11] Jensen, Kurt (1997). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Volume 2, Springer-Verlag.