Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

AN INTRODUCTION TO SYSTEMS MODELING AND SIMULATION WITH COLORED PETRI NETS

Vijay Gehlot
Carmen Nigro

Villanova University
Center of Excellence in Enterprise Technology (CEET)
Department of Computing Sciences
800 Lancaster Avenue
Villanova, PA 19085, U.S.A.

ABSTRACT

Petri Nets provide a graphical notation for modeling systems and performing analysis. Colored Petri Nets (CPNs) combine the strengths of ordinary Petri Nets with a high level programming language, making them more suitable for modeling large systems. A CPN model is an executable representation of a system that can be analyzed through simulation. CPN models are built using CPN Tools, a graphical software tool and interface used to create, edit, simulate, and analyze models. This tutorial is meant to introduce the reader to the vocabulary and constructs of CPNs and illustrate the use of CPN Tools in creating and simulating models by means of a familiar, simple example. In particular, we show how to create a CPN model of the call center example presented by White and Ingalls in their tutorial *Introduction to Simulation*.

1 INTRODUCTION

Colored Petri Nets (CPNs) provide a modeling framework suitable for simulating distributed and concurrent processes with both synchronous and asynchronous communication. They are useful in modeling both nondeterministic and stochastic processes as well. Simulation is experimentation with a model of a system (White and Ingalls 2009). A CPN model is an executable representation of a system consisting of the states of the system and the events or transitions that cause the system to change its state. Through simulations of a CPN model, it is possible to examine and explore various scenarios and behaviors of a system. The relatively small basic vocabulary of CPNs allows for great flexibility in modeling a wide variety of application domains, including communication protocols, data networks, distributed algorithms, and embedded systems (Peterson 1981, Jensen and Kristensen 2009).

CPNs combine the graphical components of ordinary Petri Nets with the strengths of a high level programming language, making them suitable for modeling complex systems (Jensen, Kristensen, and Wells 2007). Petri Nets provide the foundation for modeling concurrency, communication, and synchronization, while a high level programming language provides the foundation for the definition of data types and the manipulations of data values. The CPN language allows the model to be represented as a set of modules, allowing complex nets (and systems) to be represented in a hierarchical manner.

CPNs allow for the creation of both timed and untimed models. Simulations of untimed models are usually used to validate the logical correctness of a system, while simulations of timed models are used to evaluate the performance of a system. Time plays an important role in the performance analysis of concurrent systems.

CPN models can be constructed using CPN Tools, a graphical software tool used to create, edit, simulate, and analyze models. CPN Tools has a graphical editor that allows the user to create and arrange the various Petri Net components. One of the key features of CPN Tools is that it visually divides the hierarchical components of a CPN, enhancing its readability without affecting the execution of the model. CPN Tools also provides a monitoring facility to conduct performance analysis of a system. In addition, unlike traditional discrete event systems, CPNs allow for state space based exploration and analysis, which is complementary to pure simulation based analysis. State space analysis can be used to detect system properties such as the absence of deadlocks.

This tutorial is intended to give the reader a basic introduction to CPNs, as well as CPN Tools, and how they can be used in modeling and simulation. We will introduce the key concepts and vocabulary of CPNs by means of simple illustrative examples. We then present the steps and creation of a CPN model of the call center example discussed in White and Ingalls (2009). This tutorial requires no prior knowledge of Petri Nets, however, a basic understanding of modeling and simulation is assumed. Its emphasis is on the practical use of CPNs and the CPN Tools environment in the simulation of various systems. For sake of brevity, this paper does not discuss state-space based analysis. The rest of the paper will be organized as follows. In section 2 we introduce the details of Petri Nets and CPNs for the purposes of modeling and simulation. In section 3 we describe the CPN model of the call center example. Section 4 will be devoted to the simulation and generation of statistics for the call center example. Details of interacting with the CPN development environment (CPN Tools) are given in section 5.

2 ELEMENTS OF PETRI NETS AND COLORED PETRI NETS

CPNs are an extension of ordinary Petri Nets. Petri Nets can be used to model a wide range of various systems. This section introduces the key components of CPNs and the underlying Petri Nets formalism.

2.1 Places, Transitions, and Arcs

Places, transitions, and arcs are the basic Petri Net components. A Petri Net can be thought of as a bipartite graph consisting of two types of nodes, places and transitions. Places are displayed pictorially as circles (or ovals) and transitions are displayed as rectangles. An example Petri Net consisting of two places P1 and P2 and one transition T2 is shown in Figure 1. Note that arcs connect a place to a transition or a transition to a place, but they do not connect two places or two transitions.

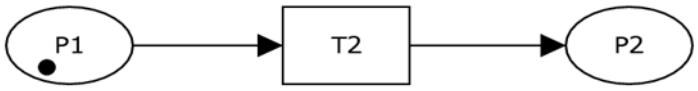

Figure 1: Basic Petri Net configuration

The interpretation of places and transitions depends on the system being modeled. Places could represent resource status or operations. Arcs often represent the flow of data or resources. Transitions could represent the start/finish of processes. In terms of simulations, transitions can be used to model both *activities* and *events*. Activities can be thought of as the processes and logic of the system, while events occur at a single point in time and cause a change in the state of a system (White and Ingalls 2009). In fact, a transition may act as a super-process consisting of many sub-processes. This is where hierarchical nets come into play (which we will explain later). Often transitions can change the state of a net through the manipulation of *tokens* via the *firing rule* which is explained next.

2.2 Tokens and the Firing Rule

Tokens are used to indicate whether certain conditions have been met so that a transition may fire. The *marking* of the net is defined by the distrubution of tokens, which are contained by places. In a pictorial

representation, tokens are indicated by black dots. For example, the net of Figure 1 has one token in place *P1*. Arcs may be labeled by weights which can specify the number of tokens needed for a transition to fire. If omitted, a value of one is assumed which is the case for the net in Figure 1. Access to tokens is controlled by the structure of the net. When the conditions on the incoming arc of a transition are met by its preplaces, the transition is said to be *enabled*. When a transition is enabled it may be fired. The firing of a transition causes a redistribution of tokens, creating a new marking (or state of the system). The number of tokens removed from a place is determined by the label on its outgoing arc and the number of tokens added to a place is determined by the label on its incoming arc when the associated transition fires. As an example, consider the marked net in Figure 2. The net represents the chemical reaction necessary to make water. The firing of transition *MoleculesReact* removes two tokens from place *Hydrogen* and one token from place *Oxygen* and adds two tokens in the place *Water*, thus capturing the reaction $2H_2 + O_2 \rightarrow 2H_2O$.

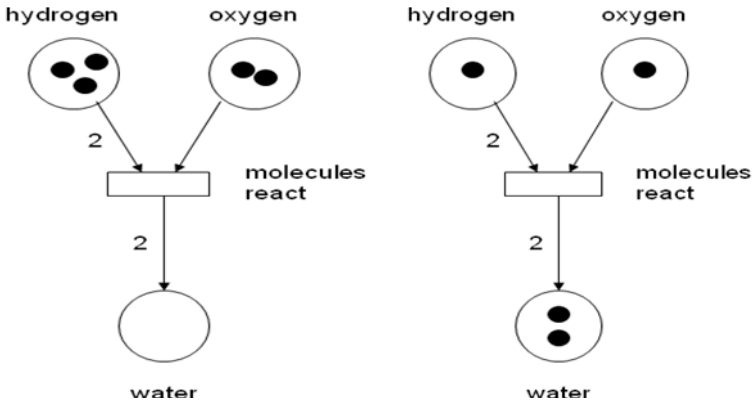

Figure 2: Marked net before (left) and after (right) firing of transition

Petri Nets provide a framework for modeling distributed and concurrent systems systems with synchronous and asynchronous communications and resource sharing.. Petri Nets also allow for mutual exclusion which can be used in representing resource constrains. This flexibility of Petri Nets renders them useful for modeling a variety of systems and situations. Figure 3 below shows some net configurations that can be useful in modeling a variety of systems.

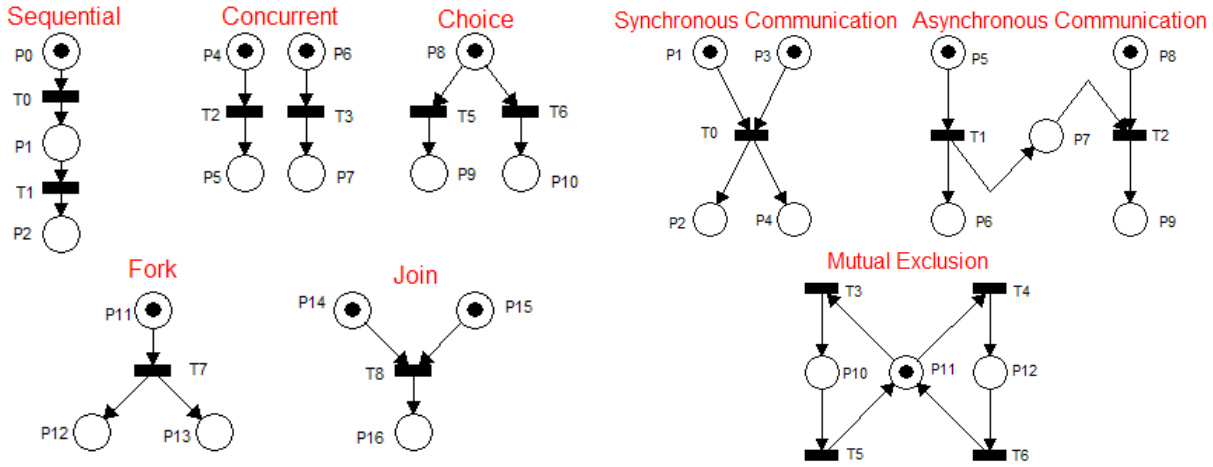

Figure 3: Some useful net configurations

2.3 Colored Petri Net Extension

Colored Petri Nets (CPNs) extend the vocabulary of ordinary Petri Nets and add features that make them suitable for modeling large systems. CPNs combine the strengths of ordinary Petri Nets with the strengths of a high-level programming language called CPN ML which is based on the functional language SML (Ullman 1998). Petri Nets provide the primitives for process interaction, while the programming language provides the primitives for the definition of data types and the manipulations of data values. Thus in a CPN model, tokens can be coded as data values of a rich set of types (called color sets) and arc incriptions can be computed expressions and not just constants. Figure 4 displays the CPN equivalent of the water example from Figure 2.

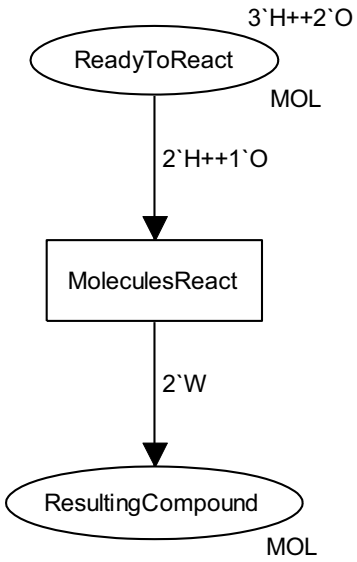

Figure 4: CPN model of the water example

In this example, the initial state *ReadyToReact* contains three `H` tokens and and two `O` tokens, which are labeled on the top right of the place. The ++ operator is used to represent a multi-set union operation. The weight on the arc signifies that two tokens of hydrogen and one token of oxygen are needed for the *MoleculesReact* transition to be enabled and fire. Once the transition is fired, the tokens are taken from the preplace and two `W`, or water, tokens are put into the *ResultingCompound* place.

In an ordinary Petri Net, tokens are indistinguishable, while in CPN all tokens are assigned a value. The values of tokens have a type and can be manipulated through the use of a high level programming language. For each state of a net, the type of contents must be specified. In Figure 2, each place has the type of token (`MOL`) labelled on the bottom right. `MOL` is a datatype which can be defined using the high level language CPN ML. In CPN ML all datatypes are referred to as color sets. We can define a color set as follows:

```
colset MOL = with H | O | W;
```

The above declaration indicates that the MOL color set (an enumeration type) can be made up of values *H*, *O*, and *W*. Note that since tokens are allowed to have separate identities; we no longer need to keep the place for Oxygen separate from place for Hydrogen. Even in this simple example, we see that CPN models tend to be much more compact when compared to their Petri Net counterpart.

In simulations tokens can be used to specify both *entities* and *attributes.* Entities effect the changes in the state of the system, while attributes are characteristics of a given entity (White and Ingalls 2009). Entities can be declared as a product of several attributes. An example of defining a compound type color set is shown below:

```
colset EMPLOYEE = ID*RATEOFPAY;
```

The above declaration describes an employee data type, consisting of an identification and pay-rate. ID and RATEOFPAY are also color sets. ID, in this case, is made up of a string and RATEOFPAY is made up of an integer. Their declarations are as follows:

```
colset ID = STRING;
colset RATEOFPAY = INT;
```

CPNs provide a rich set of constructs to create many different compound types including enumerated, subrange, product, record, union, and list types. Tokens can also be used to represent resources. In a simulation, resources can be any good that is available in a limited quantity. Resources may be shared amongst or consumed by entities. The flow and consumption or creation of resources is represented by the structure of the net.

2.4 Inputs, Outputs, and State

The input to a CPN simulation is the initial marking of the net. CPNs provide a localized view of a system. Inputs change the individual states of the system. Transitions change individual states as well as the overall state of the system. The combination of the states, or the final marking of the net, can be considered the output of the simulation.

2.5 Variables

CPN allows for variables in various net inscription expressions. All variables must be declared with their type (color set). For example,

```
var count: INT;
```

declares a variable named count of type INT. Variables can be of simple or compound types. Variables are bound to values from their declared color set by the simulator as it attempts to determine if a transition is enabled. The scope of a variable is local to the transition and there can be multiple bindings simultaneously active on different transitions. These bindings can exist simultaneously because they have different scopes. The extent of a CPN variable binding is the firing of a particular transition. In addition, CPN allows for global variables (called ref variables) as well as constants. The former are declared using the `globref` keyword whereas the latter are declared using the `val` keyword.

2.6 Random Number Generator

CPN Tools utilizes ML's random number generator to add variability to a model. This can be particularly useful in simulations to represent delays of an unknown period of time. CPN Tools provides several different random distribution functions including discrete, exponential, and uniform. A complete list can be found at:
http://wiki.daimi.au.dk/cpntools-help/random_distribution_funct.wiki.
For example, the role of a die may be represented by a call to the discrete function:

```
discrete(1,6);
```

The above function call can be used to generate a random integer between one and six.

2.7 Clock and Calendar

In CPN simulator the *clock* is represented as an integer counter. The internal type of this counter allows arbitrarily large integer values and is not restricted to, say, a 32-bit representation. Depending on the system being modeled, the clock value may be interpreted as milliseconds, or seconds, or minutes, etc. The *calendar* is represented by the distribution of tokens with their time stamps and associated transitions in a timed model. The time stamp represents the earliest possible time that a specific token may be consumed by a transition. This may depend on the execution of other events as well. Since, it is possible for models to be untimed, one may think of calendar as a list of events that can occur in some order that is not necessarily determined by time.

2.8 Statistics Collectors

CPNs provide a very extensive set of facilities to monitor a net during execution and extract relevant data (Lindstrøm and Wells 2002). In its simplest form, one can record every transition that fires and associated bindings of variables together with step and time. In most cases, however, we need to associate monitors selectively to a net or a subnet or a part thereof. CPNs include four categories of monitors:

- **Breakpoint monitors:** useful for stopping simulation based on certain conditions or criteria.
- **Data collector monitors:** useful for extracting numerical data during simulations. The numerical data is used in automatically calculating statistics, and the data is also saved in data collector log files for other processing. There are several predefined data collection monitors available.
- **Write-in-file monitors:** allows extraction and recording of generic (not necessarily numeric) data.
- **User-defined monitors:** allows creation of custom data collection monitors.

The numerical data that is extracted is used to calculate statistics. The statistics that are calculated for a particular data collector will be either *untimed statistics* or *timed statistics* (that is, time-dependent weighed statistics). The statistics that are computed and can be accessed from each data collector monitor are: count (number of data observations), minimum, maximum, sum, average, confidence intervals for average, variance, standard deviation, sum of squares, sum of squares of deviation, first value observed, and last value observed. There is also a facility to run any number of simulation replications, collect data, and calculate, among other values, 90%, 95%, and 99% confidence intervals for averages.

2.9 Advantages

CPNs provide a unified approach for analysis of both functional/logical properties as well as performance properties through their support for both timed and untimed activities within a single formalism. Thus one does not have to create two separate models to carry out such analyses. Furthermore, unlike many discrete event simulation systems, CPNs provide a set of state space methods for verification of system properties. The state space approach complements analysis that can be performed based on pure simulation. For example, the state space approach can be used to identify system deadlock states.

From a practical applications point of view, CPNs support a mechanism of modules for construction of large system models in a hierarchical manner. The hierarchy and module concept of CPNs permit different levels of abstraction that are inherent in most complex systems. The graphical representation makes it easy to see the basic structure of a complex CPN model and understand the interaction of individual sub-components. CPNs can be used in modeling a wide variety of application domains including communication protocols, data networks, distributed algorithms, embedded systems, business process and workflow, manufacturing systems, and agent systems (Jensen et al. 2007).

We illustrate timed CPNs and the hierarchical construction in the next section with an example. A list of large-scale practical models created using the CPNs is available at http://www.daimi.au.dk/CPnets/intro/example_indu.html.

3 CPN MODEL OF THE CALL CENTER EXAMPLE

To illustrate the details of model creation using CPNs, we consider the call center example from White and Ingalls (2009). The description of this system as given in the aforementioned paper is as follows:

"...Arriving calls first connect to a telephone switch. If the number of calls currently on hold is greater than ten, the caller receives a busy signal and immediately hangs up. Other-wise, the call is delivered to an automated interactive voice response (IVR) unit. The caller is asked to, "Dial one for car-stereo products; dial two for all other products" and the call is routed accordingly. The call then waits in the appropriate queue (listening to classic rock) until the first sales representative servicing the identified

product type becomes available. Finally the call is processed and the caller hangs up…."

Figure 5 below from the same paper shows the logic flow of the call center example.

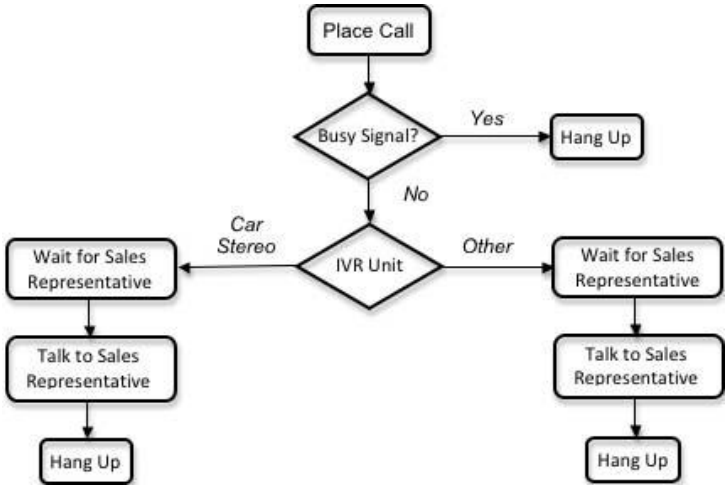

Figure 5: Call center example from White and Ingalls (2009)

Our approach is to use hierarchical CPNs to model this example. Creation of hierarchical nets is based on the idea that a transition can be replaced or substituted by a (sub) net that details the activities underlying the associated transition. Such transitions are called *substitution transitions* in the CPN parlance. Pictorially, a substitution transition is drawn with double rectangles. At an abstract level, we can view a call center as comprising of two sets of activities, namely, next call and process call. This view is captured and depicted by the top level net of our model shown in Figure **6**.

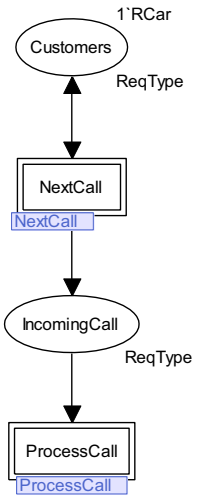

Figure 6: Top net of the CPN hierarchical model of the call center

The color set *ReqType* labeling the two places denotes the types of incoming requests. For the call center example there are only two requests: car stereo and other. Since no other attributes need to be represented with each request, the color set ReqType has been defined as an enumerated type. Furthermore since the requests need to carry a timestamp, it is declared as a *timed color set* as follows:

```
colset ReqType = with RCar| ROther timed;
```

The initial marking inscription *1`RCar* indicates that the system will start with an initial car stereo call. In CPN it is possible to read initialization values from a file or network. Details of activities associated with the substitution transition *NextCall* are shown in Figure 7.

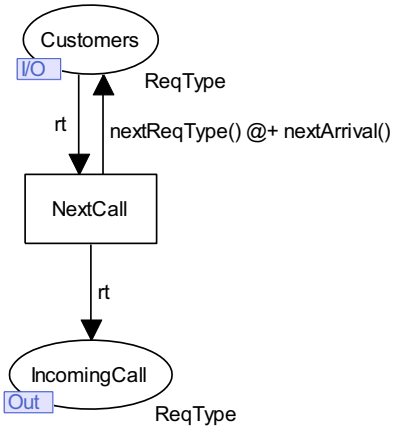

Figure 7: Details of next call generation

The example in White and Ingalls (2009) gives formulas for computing various random values. These are defined in our CPN models as the following functions:

```
fun rollTwoDice() = discrete(1,6) + discrete(1,6);
fun nextArrival() = round (real(rollTwoDice()) * 0.333);
fun ivrDelay() = round (real(rollTwoDice()) * 0.3);
fun procDelayCar() = rollTwoDice() * 2;
fun procDelayOther() = rollTwoDice();
fun nextReqType() = if discrete(1,10) <= 4 then RCar else ROther;
```

Thus, when transition *NextCall* fires, it removes a token representing next call and adds a token returned by function `nextReqType()` with a time stamp given by adding the `nextArrival()` delay to the current simulation clock (denoted by `@+` operator).

Figure 8 shows the subnet (or module) associated with the substitution transition *ProcessCall*. The net configuration clearly shows that there are three choices associated with an incoming call: *HangUp*, *Process CarRequest* or *ProcessOtherRequest*. To simplify the picture a bit, we have omitted the IVR Unit stage and factored in the delay associated with IVR Unit into the processing time for a request.

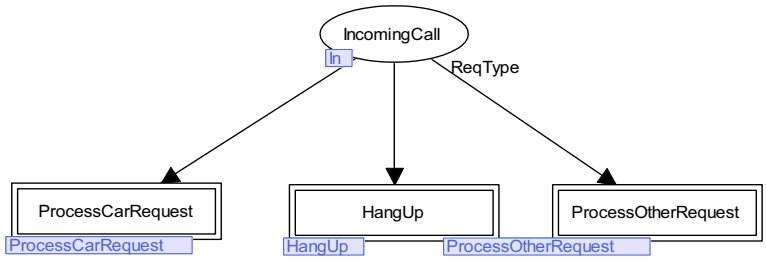

Figure 8: Subnet (module) for processing calls.

Next we detail the activities associated with module *HangUp* and *ProcessCarRequest*. We skip the details of *ProcessOtherRequest* since, in this example, it is similar to *ProcessCarRequest* except for the values of random delays and number of service representatives. The subnet detailing the hang up process is shown in Figure 9.

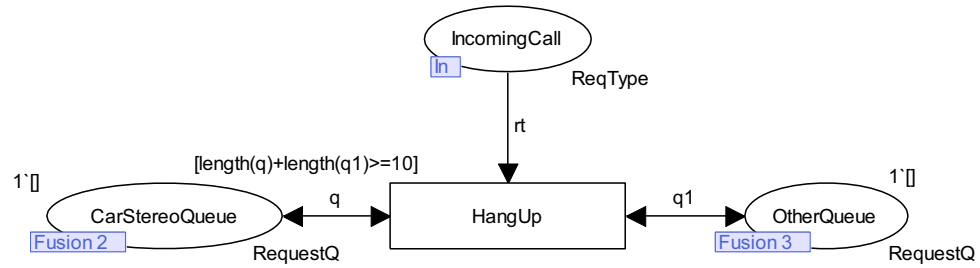

Figure 9: Subnet for hangup process

As specified in the paper, if there are already 10 calls waiting to be serviced, a busy signal is issued and the caller hangs up. This condition is checked by the guard `[length(q)+length(q1)>=10]` associated with the transition *HangUp*. CPN firing rules guarantee that this transition will not fire if the condition is not met. Figure **10** shows the subnet associated with the *ProcessCarRequest* transition.

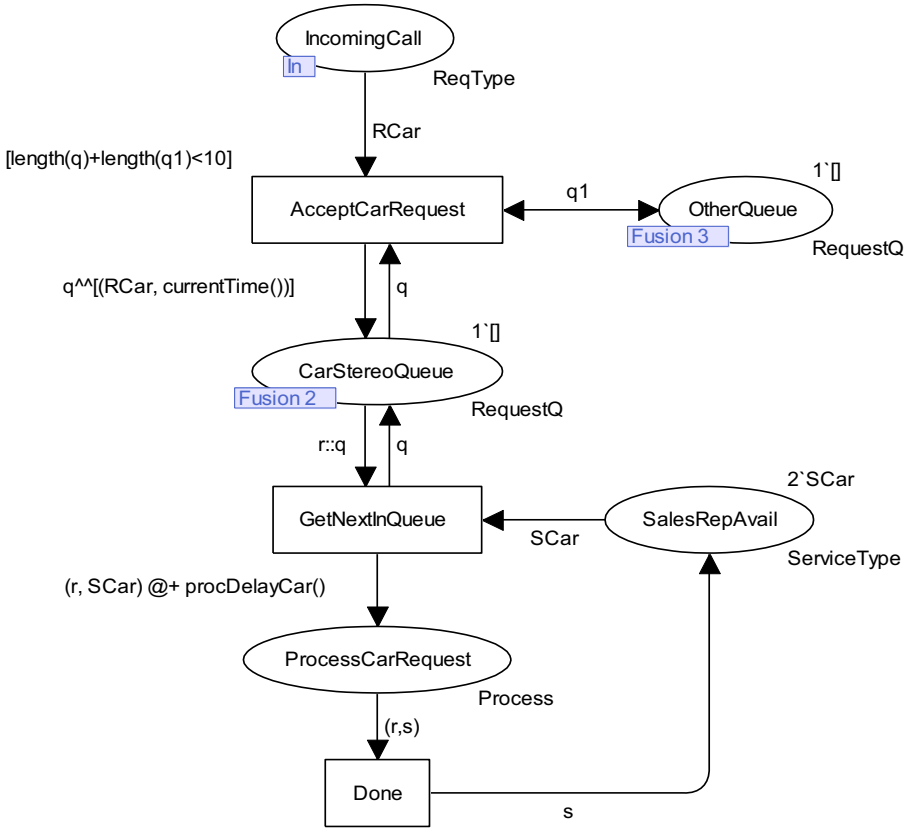

Figure 10: Subnet detailing steps of processing car stereo request

A call is accepted if it is a request for car stereo (denoted by arc inscription RCar) on the outgoing arc of *IncomingCall* place and the maximum on hold limit is not reached, that is, `length(q)+length(q1)<10`. If this condition is met, the transition fires and the new request with its start time is added to the *CarStereoQueue*. This is achieved by the arc inscription `q^^[(RCar, currentTime()]` of the incoming arc to *CarStereoQueue* place. If there is a sales representative available (denoted by arc inscription *SCar* on the arc from place *SalesRepAvail* to transition *NextIn-*

Queue) the call from the front of the queue, if any, is removed and added as a token representing a pair consisting of the request and the representative handling the request. The firing rules of CPN ascertain that a call cannot be serviced if, for example, the place *SalesRepAvail* is empty, that is, both representatives are busy handling other call*s*. This token also receives a timestamp that represents a processing delay as given by the function `procDelayCar()`. When the simulation clock advances to a value that makes this token available, the transition *Don*e gets enabled and fires, and the representative (as a resource) is returned to the available pool.

4 CALL CENTER MODEL SIMULATION AND OUTPUT STATISTICS

CPN supports automatic as well as interactive simulation. The interactive mode also allows the user to pick a transition as well as variable bindings. This mode is useful for debugging the model. In the automatic mode, the simulation can be run in either play mode or fast forward mode. In the former, after each step of execution, various markings are displayed whereas in the latter mode no intermediate markings are shown until the end of simulation. End of a simulation run is determined by simulation stop criteria which could be a number of steps, or the clock reaching a certain value or other conditions as specified by a breakpoint monitor. Figure 11 shows an intermediate marking for the subnet associated with processing a car stereo request.

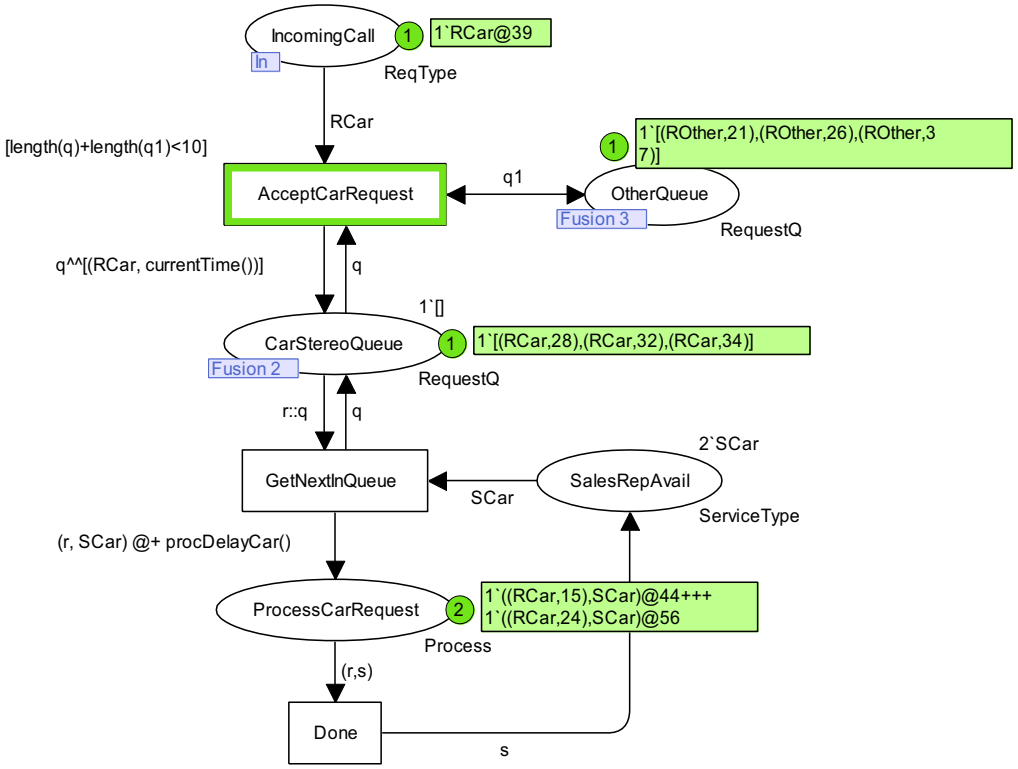

Figure 11: An intermediate marking during interactive simulation

In this marking, we see that the next incoming call is for a car stereo with time stamp 39. The simulation clock (not shown) is at 39. Thus this token is available and the associated call will be accepted and queued since there are only three calls waiting in the car queue and (coincidentally) only three calls waiting in the other queue and therefore the busy signal will not be generated. This fact is indicated by the green highlight on the transition *AcceptCarRequest* which means this transition is ready to fire next.

We associated several monitors to collect data and compute statistics from this model. For example, there is a *Count Transition Occurrences* monitor associated with transition *AcceptCarRequest* that keeps

track of how many times this transition fires. This number tells us the number of car stereo requests that were received during simulation. Similarly there is a *Count Transition Occurrences* monitor associated with transition *Done* that keeps track of how many car stereo requests were served during simulation. A List Length monitor is associated with place *CarStereoQueue* that is used in computing average queue length. When transition *GetNextInQueue* fires, a user defined data collector monitor computes the wait time by subtracting the start time from the current simulation clock value. Table 1 and Table 2 summarize the various statistics that were automatically computed for a sample simulation run consisting of 1000 steps with a simulation clock end reading of 845.

Table 1: Time dependent statistics from a sample run

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
Car_Stereo_Queue_Size	409	0.639053	0	6
Other_Queue_Size	446	8.789349	0	10

Table 2: Untimed statistics from a sample run

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Car_Queue_Wait_Time	56	540	9.642857	0	51
Completed_Car_Calls_Count	56	56	1.000000	1	1
Completed_Other_Calls_Count	92	92	1.000000	1	1
Excessive_Wait_Car_Queue_Count	56	22	0.392857	0	1
Excessive_Wait_Other_Queue_Count	93	92	0.989247	0	1
HangUps_Count	192	192	1.000000	1	1
Incoming_Call_Count	352	352	1.000000	1	1
Other_Queue_Wait_Time	93	6940	74.623656	0	100

Note that these are auto-generated and not all statistics are meaningful in all situations. For example, for Completed_Car_Calls_Count, only the sum is meaningful. As suggested in White and Ingalls (2009), 30 simulation replications were run using the CPN built-in function `CPN'Replications.nreplications` that can be used to automatically run a given number of simulations. Confidence intervals for the average of a set of data values, assumed *independent and identically distributed (IID),* are automatically computed at the end of replication run. Table 3 summarizes a portion of this auto-generated report.

5 THE CPN TOOLS INTERFACE

CPN Tools has an intuitive graphical user interface that is useful for creating and simulating CPN models. Figure 12 contains a snapshot of the CPN Tools interface for the call center example net.

Table 3: Confidence Intervals after 30 simulation replications

Statistics							
Name	**Avrg**	**90% Half Length**	**95% Half Length**	**99% Half Length**	**StD**	**Min**	**Max**
Car_Queue_Wait_Time							
count	65.333333	3.155314	3.797892	5.118332	10.172083	43	83
max	33.400000	4.687077	5.641597	7.603052	15.110170	13	62
min	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
sum	569.633333	118.877302	143.086570	192.834517	383.235903	134	1830
avrg	8.349415	1.469671	1.768968	2.383998	4.737915	2.442623	22.317073
Car_Stereo_Queue_Size							
count	414.166667	0.710548	0.855251	1.152602	2.290661	409	418
max	4.533333	0.585914	0.705235	0.950429	1.888866	2	9
min	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg	0.715616	0.152250	0.183255	0.246969	0.490822	0.162424	2.322335
Completed_Car_Calls_Count							
count	64.266667	3.069607	3.694730	4.979303	9.895779	43	82
max	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
min	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
sum	64.266667	3.069607	3.694730	4.979303	9.895779	43	82
avrg	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000
Completed_Other_Calls_Count							
count	87.700000	0.647344	0.779176	1.050077	2.086905	84	92
max	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
min	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
sum	87.700000	0.647344	0.779176	1.050077	2.086905	84	92
avrg	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000
Other_Queue_Size							
count	437.366667	3.129075	3.766308	5.075768	10.087491	420	459
max	10.000000	0.000000	0.000000	0.000000	0.000000	10	10
min	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg	8.648859	0.165822	0.199592	0.268985	0.534577	6.996193	9.286061
Other_Queue_Wait_Time							
count	88.533333	0.670425	0.806956	1.087516	2.161311	85	93

max	103.833333	1.934730	2.328736	3.138385	6.237171	94	123
min	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
sum	6560.933333	163.475232	196.766834	265.178187	527.010432	5005	7237
avrg	74.094265	1.730942	2.083447	2.807814	5.580201	58.882353	84.151163

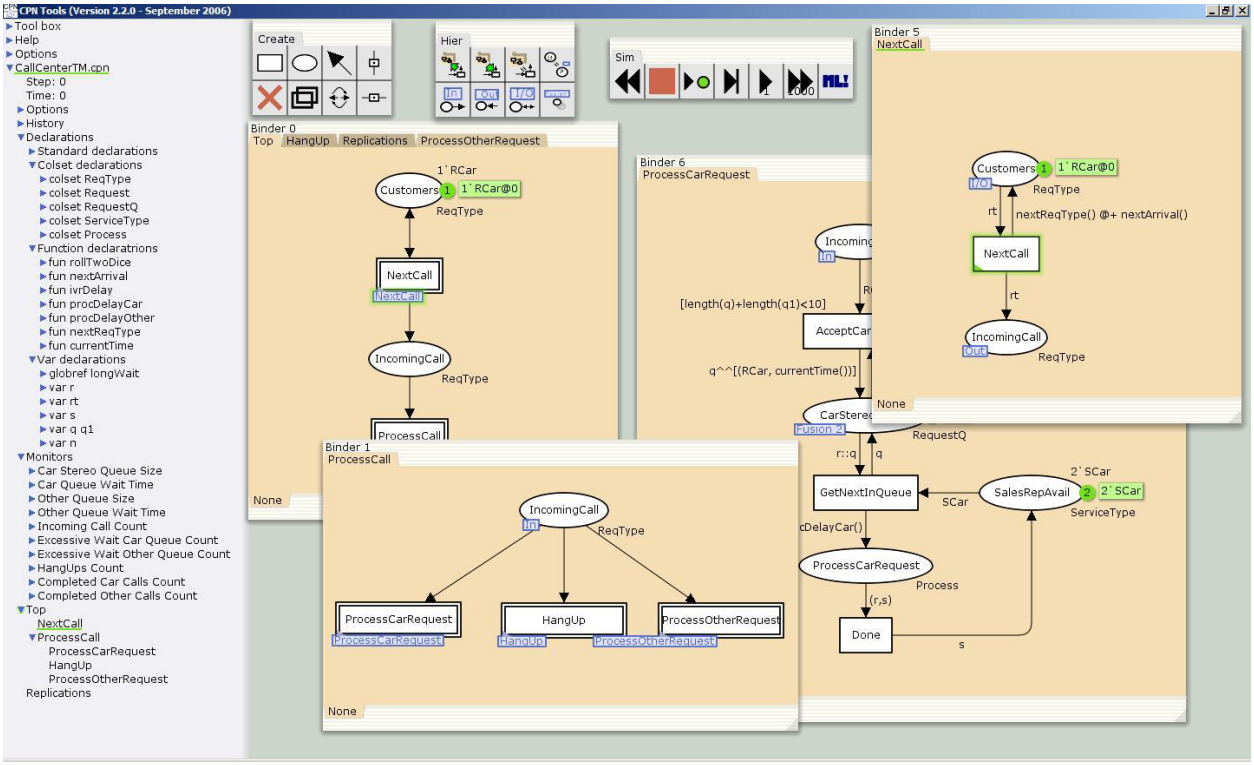

Figure 12: The CPN Tools interface

The left column is called the *index*. The index contains a hierarchical list of objects including tools and colored Petri Nets. Expanding the tool box gives a list of all the available CPN Tools. We will not go into detail for all the tools, but a complete list of all the tools and their functions can be found on the CPN Tools website (`http://wiki.daimi.au.dk/cpntools-help/cpntools-help.wiki`). Any of these tools can be dragged to the section on the right, called the *workspace*. Dragging an item into the workspace creates a view of its contents. The views of the *Create Tool*, the *Hierarchy Tool* and of the *Simulation Tool* can be seen at the top of Figure 12. Expanding a net on the index displays all the information associated with that net, including all declarations and monitors.

The Create Tools are used to create and edit the basic components of a CPN including places, transitions, and arcs. It also consists of guideline tools for easy alignment of net structures and tools for cloning and deleting net components.

The Simulation Tools are used for running interactive or automatic simulations. It contains video tape player-like controls used to manipulate the model. *Next Frame* allows the user to select a transition to fire. *Play* randomly fires enabled transitions until the end of the simulation is reached or the user hits the *stop* button. *Fast-forward* runs the simulation for a specified number of steps set by the user, while *rewind* resets the simulation to its initial state.

In CPN Tools a net is organized into *binders* that contain pages. Each page can contain a single subnet. Figure 12 shows the overall net contained in four binders. Multiple nets and binders may be open at the same time. Pages may be dragged from binder to binder without changing the execution aspect of the model. Dragging a page into the workspace will create a new binder for that page.

The page named Top is the top-level page and contains two *subpages*, each represented as a transition on the page. Each of the subpages is given a page in a binder. These subpages may have subpages of their own and so on. The various page relationships are depicted in the index via the indentation of the index entry. Hierarchy is an important feature in CPN Tools, because it allows for complex systems to be divided up and represented as several sub-models. CPN Tools supports both a bottom-up approach as well as a top-down approach for creating hierarchical net models.

6 CONCLUSIONS

The focus of this paper has been to introduce the audience to modeling and simulation with Colored Petri Nets. We presented the major concepts, vocabulary and constructs of a CPN model and their use in building executable models for system simulation. We've also explained the flexibility of CPNs and many of CPNs' important features including hierarchy, color sets, various net configurations, and both timed and untimed nets. We have also addressed the practical applications of CPNs and how their flexibility allows for their use in modeling a variety of systems including communication protocols, business processes and workflow, manufacturing systems, and embedded systems (Jensen et al. 2007). Stepping through the call center example from White and Ingalls (2009), we were able to present the modeling and execution of a CPN simulation that could be put to practical use. We illustrated the use of monitoring facilities for automatic data collection and statistics computation. Unlike many discrete event simulation tools, CPN Tools does not provide built-in facilities for the visualization of data. There is an indirect support for visualization through GNU Plot scripts that the tool is capable of generating automatically. Finally, we gave a brief introduction to CPN Tools, an intuitive software tool used to create and execute CPN models. A larger example appears in Gehlot and Hayrapetyan (2006). The paper does not focus on state-space based analysis. A high-level introduction to state-space analysis and its use can be found in Jensen, Kristensen, and Wells (2007). Details of theoretical foundations including formal definitions are described in Jensen (1994).

CPN Tools was created by the CPN Group of the University of Aarhus in Denmark. It can be obtained free of charge by agreeing to their license. The license application is available at `http://www.daimi.au.dk/~cpntools/bin/license/apply.php`.

The practical application of CPNs is contingent on automatic and interactive simulation, visualization, state space analysis, and performance analysis. All of these help in the verification and validation of a modeled system. CPN models can be used to validate both the logic of a system and its performance, saving the need to create two independent models, making CPN a powerful modeling tool.

REFERENCES

Gehlot, V., and A. Hayrapetyan. 2006. A CPN model of a SIP-based dynamic discovery protocol for webservices in a mobile environment. In *Proceedings of the 7th Workshop on Practical Use of Coloured Petri Nets (CPN'06)*, 197–216.

Jensen, K. 1994. An introduction to the theoretical aspects of coloured Petri nets. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): *A Decade of Concurrency, Lecture Notes in Computer Science* 803:230-272, Springer-Verlag.

Jensen, K., and L.M. Kristensen. 2009. *Coloured Petri Nets. Modelling and Validation of Concurrent Systems*. Springer.

Jensen, K., L.M. Kristensen, and L. Wells. 2007. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(3–4):213–254.

Lindstrøm, B., and L. Wells. 2002. Towards a Monitoring Framework for Discrete Event System Simulations. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*.

Peterson, J.L. 1981. *Petri Net Theory and the Modeling of Systems*, Prentice-Hall.

Ullman, J.D. 1998. *Elements of ML Programming,* Prentice-Hall.

White, K.P., Jr. and R.G. Ingalls. 2009. Introduction to Simulation, In *Proceedings of the 2009 Winter Simulation Conference*, eds. M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, 12-23. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

VIJAY GEHLOT is Associate Professor and Graduate Program Director in the Computing Sciences Department at Villanova University. He received a Bachelors of Engineering (Hons.) in Electrical and Electronics from Birla Institute of Technology and Science (BITS), Pilani, India, a Masters of Engineering in Automation from Indian Institute of Science (IISc), Bangalore, India, and a Ph.D. in Computer and Information Science from University of Pennsylvania (UPENN), Philadelphia, Pennsylvania. His research interest are in the area of systems modeling and analysis, formal methods, and applications of colored Petri nets. He has used CPNs to model a large scale multi-channel SOA system for the US Air Force. In addition he was also involved in creation and integration of an agent-based economic model into an asset planning tool for the US Army. He has used CPN modeling to identify patient safety issues in wireless medical device networks in hospitals. He is a member of the ACM and Sigma Xi. His email address is <vijay.gehlot@villanova.edu>.

CARMEN NIGRO is a junior in the Bachelor of Science in Computer Science program at Villanova University. He has been accepted into the five year BS/MS program at Villanova and plans to work towards a Masters degree at Villanova. His research interest include modeling and simulation with CPNs. He has been actively involved in a project for the US Army to create a Drupal-based asset planning tool and integrate it with CPN Tools for modeling and analysis. His email address is <carmen.nigro@villanova.edu>.