Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

COMPOSABILITY VERIFICATION OF REAL TIME SYSTEM MODELS USING TIMED COLORED PETRI NETS

Imran Mahmood
Rassul Ayani
Vladimir Vlassov
KTH Royal Institute of Technology
Isafjordgatan 39,
Stockholm, 16440-Kista, SWEDEN

Farshad Moradi
Swedish Defense Research Agency (FOI)
Gullfossgatan 6
Stockholm, 164 90 SWEDEN

ABSTRACT

The discipline of component-based modeling and simulation offers promising gains including reduction in development cost, time, and system complexity. It also promotes (re)use of modular components to build complex simulations. Many important issues in this area have been addressed, but *composability verification* is still considered a daunting challenge. In our observation most of the component modeling frameworks possess weak built-in support for the composability verification, which is required to guarantee the correctness of the dynamic (behavioral) and temporal aspects of the composition. In this paper we stage a practical approach to alleviate some of the challenges in composability verification and propose a process to verify composability of real-time system models. We emphasize on *dynamic semantic level* and present our approach using *Colored Petri Nets* and *State-Space analysis*. We also present a Field Artillery model as an example of real-time system and explain how our approach verifies model composability.

1 INTRODUCTION

The Modeling and Simulation (M&S) community has been conducting research on methods and technologies to construct complex simulation systems by combining new or reusing existing simulation components. This paradigm of component-based modeling and simulation has gained growing motivation due to its promising gains including reduction in development cost, time, and the system complexity. It follows the principle of *modularity* which essentially helps to master the *complexity of reality by decomposing it into parts* (Hofmann 2003)and by enabling the designer to (re)use appropriate parts for different purposes.

Composability is the capability to select and assemble components in various combinations (meaningfully) to satisfy specific user requirements (Petty and Weisel 2004). It is an important quality characteristic of the M&S discipline, yet difficult to achieve (Balci, Arthur and Ormsby 2011), (Davis and Anderson 2003). This is mainly due to the underlying intricacies and substantive subtleties of the components. Composability is a property of the models, as it essentially contends with the alignment of issues on the modeling level (Tolk 2010), where it is viewed as creation of complex models from a collection of modular components, which might themselves be the abstraction of subsystems. Composability essentially relies on a suitable component modeling framework that must provide accurate reasoning of correctness and the ability to leverage certain component standard. One such standard that has been developed for M&S to support composability is the Base Object Model (BOM) (Gustavson 2005), which is a SISO standard. Composability is further divided into different sublevels, as discussed in (Moradi, et al. 2007) & (Tolk 2010).

In this paper, our focus is centered on the correctness of composability at the *Dynamic Semantic level*, which is a necessary condition for the credibility of overall composability. Dynamic Semantic Composability implies that the components are dynamically consistent, i.e., they have correct behavior, necessary to reach the desired goals and subsequently satisfy user requirements. In essence, a set of components can possibly fit together (*syntactically*), and their communication is meaningful and understood (*semantical-*

ly), but unless all components preserve essential behavior (*dynamically*), in order to reach the desired composition goals, the correctness of the composed model cannot be certified. We further elaborate that correctness of behavioral composability relies on two factors: firstly each component is at the right state while interacting with the others, and secondly the composition should satisfy required behavioral properties, as prescribed in the requirement specifications

In M&S, verification is concerned with building the model right. It is typically defined as a process of determining whether the model has been implemented correctly (O. Balci 1997) and whether it is consistent with its specifications (Petty 2008). In principle, verification is concerned with the accuracy of transforming the model's requirements into a conceptual model and the conceptual model into an executable model (Petty 2008). We postulate that model's requirements are identified by means of requirements specification which includes a set of verification goals, listed in terms of desired system behavior properties such as *deadlock freedom*, *livelock freedom*, *mutual exclusion* and *fairness*. In dynamic semantic composability verification, we show that the composed model satisfies its requirement specification.

Systems where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced and within given time bounds, are known as *Real-Time* systems (Olderog and Dierks 2008). When models of such systems are composed, they may also require having certain time properties that should be satisfied for correct composability.

Various approaches have been suggested concerning the model verification of real-time systems. A formal model of hierarchical time system is presented in (David, Möller and Yi Wang) and adjoined with an 'Uppaal tool' for formal verification. Another interesting approach uses DEVS for real-time system development, and transforms it to Timed Automata for verification using Uppaal tool (Saadawi and Wainer 2009). A rather different approach focuses on the formal validation of semantic composability of time based systems (Szabo, Teo and See 2009).

In this paper, we present composability verification of real time system models, where time constraints are defined as requirement specification properties and their behavior is evaluated to guarantee response within the required time constraints. In our approach, we suggest to use BOM as a conceptual modeling standard, and Colored Petri Nets (CPN) as an executable modeling framework. We propose a formal Time CPN-based component model, which is used for implementation and execution of BOM components. After implementing BOM components (using our automatic transformation method) the generated CPN-component models are composed and subjected to the verification process for the evaluation of composability at dynamic semantic level. A verified composition of CPN-component models asserts that the composability of their respective conceptual model is correct, with respect to the given requirement specifications. We also present a Field Artillery model as an example to illustrate how our approach verifies a composed model.

The rest of the paper is organized as follows: Section 2 covers basic definitions and concepts used in this paper. Section 3 furnishes the details of our composability verification process. In Section 4 we discuss a case study of a Field Artillery model to explain our approach, whereas section 5 frames summary and conclusion.

2 DEFINATIONS AND BASIC CONCEPTS

In this section we briefly discuss some essential concepts that are used later in this paper.

2.1 Base Object Model

Conceptual modelling is an iterative process of abstracting an appropriate simplification from a real (or proposed) system, to form a model (Robinson, et al. 2010) according to the given requirements and modeling objectives.

The SISO standard BOM (SISO-I 2006) is defined as, "*a piece part of a conceptual model, composed of a group of interrelated elements, which can be used as a building block in the development and extension of simulations and simulation environments*" (Gustavson 2005). BOM includes the aspects of a conceptual model that captures static descriptions of elements abstracted from the real system *(simuland)*, described in terms of conceptual *entities* and *events*, and contains information on how these elements interact with each other in terms of *Patterns of Interplay* and *state-machines*. Entities and Events represent data about the real world objects and their interaction, whereas the pattern of interplay and state-machine collectively represents the dynamic behavior of the component. In this paper, we harness the capability of BOM as a conceptual modeling framework, because it provides a component standard as a basis of model documentation; gives guidelines for the further development of the executable model and helps determine the appropriateness of the model or its parts for model reuse; and most importantly, it strongly supports composability.

2.2 Colored Petri Nets

A conceptual model is by definition vague and ambiguous. It is then refined into a more concrete executable model. The process of model design involves the development and refinement of this vague and ambiguous model and creating the model code (Fishwick 1995), we refer to it as an *executable model*. In this paper, we incorporate Colored Petri Nets formalism (developed at the University of Aarhus), as an executable modeling framework, focusing on its Time extension in particular and propose to utilize its strength by implementing BOM based conceptual model into a Timed-CPN based executable model.

CPN is a general purpose discrete event graphical language for constructing models of concurrent systems and analyzing their properties. It combines the capabilities of Petri nets, as a foundation of the graphical notation, and a programming language CPN ML (an extension of Standard ML) that provides the primitives for the definition of data types and for specifying functional routines. TCPN is an extension to CPN, in which tokens can carry *timestamps* in addition to the token color, which implies that the *marking* of a place where the tokens carry timestamps become timed multi-sets. Also, the model has a global clock, representing model time. The distribution of tokens on the places, together with their timestamps and the value of the global clock, is called a *timed marking.* For detailed explanation of the concepts of Timed CPN, interested readers are recommended to consult (Jensen and Kristensen 2009).

'CPN Tools' is a modeling and execution environment based on CPN language, and is used for the editing, simulation, state space analysis, and performance analysis of CPN models. It also comes along with a bundled simulator that handles the execution of both untimed and timed nets. The most important features of CPN tool from our point of view are hierarchal CPN modeling and the generation and analysis of state spaces. Hierarchal CPN modeling offers modular development. CPN model can be organized as a set of modules; where modules can be seen as 'components' and can be composed in a separate model. CPN tools offer facility to construct hierarchal CPN models, by replacing an entire CPN model with a substitute transition that can be connected to a main model. In (Mahmood, et al. 2012) and in this paper, we have utilized this feature, and develop a CPN based hierarchal "Component Model", for implementation and execution of a conceptual model.

Some of the important CPN related terms, that we use in this paper are briefly described as follows:

- A *marking* is a function M that maps each place $p \in P$ into a multi-set of tokens $M(p) \in C(p)MS$.
  CPN places represent the state of the modeled system and place can be marked with one or more tokens, and each token has a data value attached to it. This data value is called the token color. It is the number of tokens and the token colors on the individual places which together represent the state of the system. This is called a marking of the CPN model.
- The *initial marking* M_0 is defined by the initial data value of tokens for all places $p \in P$ at $M_0(p)$.
- The *variables* of a transition t are the arc variables that appear on arcs connected to t.
- A *binding* of a transition t is a function b that maps each variable $v \in Var(t)$ into a value $b(v) \in Type[v]$. The set of all bindings for a transition t is denoted B(t).
- A *binding element* is a pair (t,b) such that $t \in T$ and $b \in B(t)$.
- For a binding element (t,b) to be *enabled* in a marking M, there must be sufficient tokens on the input places of the transition and the guard (exit condition) should be satisfied.

- When an enabled binding element (t,b) *occurs*, it removes tokens from the input places of transition t and adds tokens to the output places of t. For details we refer to (Jensen and Kristensen 2009).

2.3 State Space Analysis

State space analysis is one of the most prominent approach for conducting formal analysis and verification. The basic idea in this approach is to calculate all possible system states and represent them as vertices in a directed graph and represent the transitions of one state to another state by directed edges.

In theory a state space is a directed graph where we have a node for each reachable marking and an arc for each occurring binding element. There is an arc labeled with a binding element (t,b) from a node representing a marking M_i to a node representing a marking M_{i+1} if and only if the binding (t,b) is enabled in M_i and the occurrence of (t,b) in M_i leads to the marking M_{i+1}.

A constructed state space can answer a large set of analysis and verification questions concerning the behavior of the system such as absence of deadlocks, the possibility of being able to reach good state(s), and never reach bad state(s) and the guarantee of reaching goal state(s). A step by step tour of state space analysis using CPN tools can be located at (Jensen, Christense and Kristensen 2006).

The main advantages of state space methods is that they can provide counter examples and reasoning as to why an expected property does not hold. Furthermore, the automatic calculation and generation of state-space provides an ease of use, due to the fact that the computer tool hide a large portion of the underlying complex mathematics from the user, who is only required to formulate the property which is to be investigated and a suitable query function to evaluate it (Kristensen 2000).

The main disadvantage of using state spaces is the state explosion problem. Even relatively small systems may have an astronomical or even infinite number of reachable states. This problem escalates severely, when the models include Time. A lot of effort has been invested in the development of reduction methods to alleviate this problem. Reduction methods avoid representing the entire state space of the system or represent the state space in a compact form. The reduction is done in such a way that properties of the system are preserved and can still be derived from the reduced state space. However, due to the complexity and diversity in verification, there is no single reduction method which works well in all situations, therefore the choice of a reduction method, depends on the nature of the system being verified (Kristensen 2000). Some of the important reduction methods are Sweep line method (Christensen, Kristensen and Mailund 2001), Hash Compaction Method (Westergaard, et al. 2007), Symmetry Method (Elgaard July 2002) and Equivalence Method (Jensen and Kristensen 2009). The detail discussion of these methods is out of scope of this paper, however we rely on these methods, to alleviate the state explosion problem, if the model under consideration becomes large and resource intensive.

3 COMPOSABILITY VERIFICATION PROCESS

In this section we revisit our previously proposed approach of dynamic semantic composability verification (Mahmood, et al. 2012), [available here] and extend it with additional features, particularly to support the verification of real-time system models. Before we discuss these additional features, we summarize our previous contributions as follows:

We proposed a process for the verification of BOM based composed models at the dynamic semantic level. We suggested to extend the BOM components into Extended BOMs (E-BOM) using our E-BOM editor, to include state-variables and more detailed transitions, with events, guards and actions, in the model. Once a standard BOM component is extended into E-BOM, it is transformed it to our CPN based component model. We proposed an automatic transformation method to convert E-BOM into CPN model. When all components are transformed, modeler can assemble them as a composed model using CPN hierarchy tools. The resultant model is executable in CPN environment and can be analyzed using state space analysis. For the purpose of verification, we proposed to use a *Verification Template* that consists of a set of properties representing Goal states, Generic system behavioral properties and scenario centric properties as requirements specification. When the verification is performed, the composed model is said to be verified at dynamic semantic level if it satisfies all the properties in the verification template. Figure 1 illustrates the entire process.

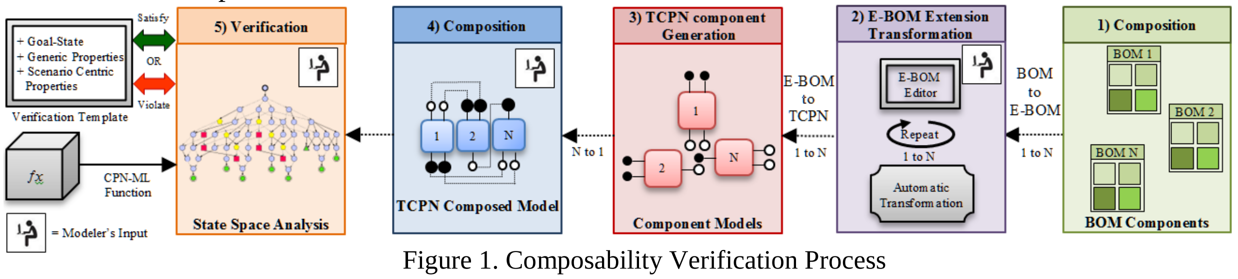

Figure 1. Composability Verification Process

In this paper, we upgrade the entire verification process by extending its different parts, described as follows.

- Since standard BOM doesn't support Time, so we present additional features in E-BOM to let the modeler specify time functions, to capture time specific system behavior.
- We upgrade our automatic transformation tool, to generate Time based CPN models
- We formally define our CPN component model and include Time functions in the formalism.
- We upgrade our CPN-Component Model by marking all the Color-Sets as "Timed", which is a requirement for their corresponding places to carry tokens with timestamps.
- We extend our Behavioral Layer and allow Time inscriptions in the transitions, which are either constant non-negative numerical values or random numbers based on an assigned probability distributions. (Time inscriptions include delay or interval functions, discussed later in this section).
- We develop a library of state space query functions for the verification of our model specific properties mentioned in verification template.
- Also we include time properties as specification in our verification template. (We provide examples in our case study section).
- Beside time related extensions, we provide modifications at different parts to improve the overall verification process.

3.1 Formal definition of CPN-based Component Model

In our model, each component has three layers and is formally defined as:

CPN-CM = (SL, BL, CL) where:

Structural layer SL = $\langle SV, \Sigma_{SV}, C_{SV}, V_{SV}, I_{SV}\rangle$ where

- SV is a set of places (that represent state-variables of the Conceptual Model)
- Σ_{SV} represents a set of color sets that represent state-variable data types
- $C_{SV} : SV \rightarrow \Sigma_{SV}$ is a function that assigns a *timed* color set to each state-variable
- V_{SV} is a finite set of typed variables, assigned to each SV such that Type[V_{SV}] = Type[SV]. (Note that these variables are CPN-ML variables that are used to transport tokens; and are different from our notions of state-variables)
- I_{SV}: SV → EXPR is an initialization function that assigns an initial value expression of type Type[SV] to each state-variable.

The structural layer represents physical attributes, properties or variables of the component. In this layer, State-Variables are defined in form of places, to model important physical changes during the execution of a component.

Behavioral layer BL = $\langle S, T, A_T, A_{SV}, A_C, G, Act, TD, TI\rangle$ where

- S is a set of places, called *states* of Type = INT (*timed*) and represent the states of the state-machine (of the respective Conceptual Model).
- T is a set of transitions (representing the events of the state-machine)

- $\mathbf{A_T}$ is the set of arcs, called *transiting-arcs* that connect a *state* $s \in S$ to a transition $t \in T$, and t to another *state* $s` \in S$, if $\mathbf{s \rightarrow t \rightarrow s`}$. All the arcs in $\mathbf{A_T}$ are assigned a common variable v of Type=INT, so that a token is transited from one state to another, to reflect a change of state in the state-machine.
- $\mathbf{A_{SV}}$ is another set of arcs, called *"sv-arcs"* that connect state-variables (of the structural layer) to the transitions. A transition t is assigned a binding b = [In | Out] with a state-variable $sv \in SV$ such that if b=In then $sv \rightarrow t$ and if b=Out then $t \rightarrow sv$. A variable v is also assigned to each arc, such that Type[v] = Type[SV(sv)].
- $\mathbf{A_C}$ is another set of arcs, called *"communication-arcs"*, that connect a communication port $cp \in CP$ (of the communication layer) to a transition t, such $cp \rightarrow t$ if t is a receive event and $t \rightarrow cp$ if t is a send event. A variable v is assigned to this arc, such that Type[v] = Type[CP(cp)].
- G : T→EXPR is a guard function that assigns a guard to a transition t such that Type[G(t)] = **Bool**
- Act: T→EXPR is an action function written in CPN-ML and assigned to a transition t. It is executed when t is fired.
- TD: T→EXPR is a time function, called *transition delay* that assigns a delay to a transition t, such that Type[TD(t)]=TIME (a non-negative integer). TD can be constant or a random function based on some probability distribution.
- TI: T→EXPR is a time function, called *transition interval,* which assigns a temporal interval to a transition t. The interval starts when t is enabled, and it remains enabled until TI expires.
  (It should be noted that the assignments of G, Act, TD and TI are optional)

  The Behavioral Layer represents state-machine behavior of a component in form of CP-Net, where each BOM state becomes a CPN place, each BOM event becomes a CPN transition, and the flow of INT type token(s), through variable v, represent the change of component's state. If specified by the modeler, the state-variables of the structural layer are connected to the transitions of this layer, to read or write a variable value when a transition occurs e.g., in Figure 2, T_0 reads *"$value_0$"* from SV_0, action Act_0, processes it and outputs the result to SV_1. (It should be noted that actions can also be used for *data marshaling,* if the types of input and output places are not the same. We define some generic data-marshaling functions for this purpose, which type-cast a token values, e.g., from $type_0$ to $type_1$ at T_0 in figure 2). The arcs can be directed in or out of the transitions, depending upon the specified bindings. The time functions *TimeFx* are responsible to control the transitions time (delay or interval), based on which the timestamps of the tokens are updated in the output places (of all layers).

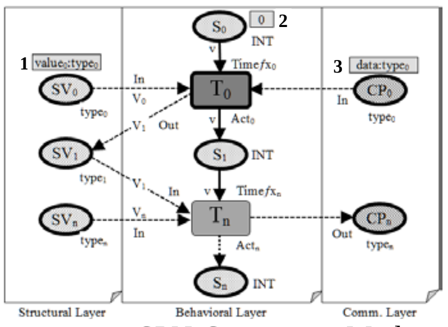

Figure 2. CPN-Component Mod-

Communication layer CL = $\langle$CP, Σ_{CP} , C_{CP}, V_{CP}, PT$\rangle$

- CP is a set of port-places (that connect to the ports of other components through sockets)
- Σ_{CP} represents a set of timed color sets that represent communication ports data types, and represent the structure of the message the parameters (as modeled in BOM).
- C_{CP} : $CP \rightarrow \Sigma_{CP}$ is a function that assigns a *timed* color set to each port-place.
- V_{CP} is a finite set of typed variables, assigned to each CP such that Type[v] ∈ Type[CP].
- PT : CP→{In, Out, I/O} is a port type function that assigns a port type to each port place.

The Communication Layer is responsible for communication with the other components. It provides interface for connecting the inputs and outputs of the components through "port places" and also provides information about the data exchange (i.e., the tokens of complex data-types that carry message parameters contents, as modeled in BOM). The transitions of behavioral layer are connected to the port places of this layer. The arc direction depends on the type of the transition (send or receive). In figure 2, each layer is initialized with required initial markings (labeled 1, 2 & 3), which fulfills the condition for the enabling of the transition T_0 and hence the component can make progress by firing T_0.

3.2 E-BOM Extension

BOM framework does fundamentally poses a satisfactory potential for effective model composability and reuse; even so it falls short of required semantics and necessary modeling characteristics of behavioral expressiveness, which are essential for modeling complex behavior of the system. Therefore we need some external specification methods that can provide essential modalities required to express additional behavioral semantics. In this paper, we propose the following extensions (in addition to the previously proposed ones). This includes state-variable bindings with the transitions and some time-functions, to capture time behavior of real-time system:

`State | Event | Guard` **`|{SV`$_{\mathtt{In}}$`}|{SV`$_{\mathtt{Out}}$`}|`** `Action | Next State` (1)

We suggest to specify transitions according to (1), where **`{SV`$_{\mathtt{In}}$`}`** means a set of input state-variables, and **`{SV`$_{\mathtt{Out}}$`}`** means a set of output state-variables, that a modeler wants to bind with this transition. It means when the system being at a state, receives or sends an event, and if the guard is satisfied, then it takes values from a set of In-bound state variables, process them in the execution of an action, output new values to the set of out-bound state-variables, and then the system will transit to the next state.

Time modalities do not exist originally in standard BOM. But when the modeling of a real-time system is under consideration, where time plays a key role, we need to provide time functions. We define two types of time functions, which can be assigned to a transition:

`Transition` **`| Time-Delay | Time-Interval`** (2)

Time-Delay means the time taken by the transition to occur. It can be a constant non-zero integer or a random function based on any probability distribution. CPN tools provide this feature of assigning time delay to a transition. *Time-Interval* means the time between the enabling of a transition and a certain specified future time, during which it will remain enabled. If other components are interacting with a component having a transition with *Time-Interval*, they can only progress if they communicate just within this interval. (We provide example in our case study section). This feature is not available in CPN. Therefore for its implementation, we propose to attach a “timer” to the transition for evaluating the lapse of the specified interval as shown in figure 3. The shaded area represents our implementation of a timer, in CPN. Whenever the place A receives token(s), it enables the transition, which when fired sets the timer to run, starting from the current model time. This transition will remain enabled and can be fired multiple times, until TI is reached.

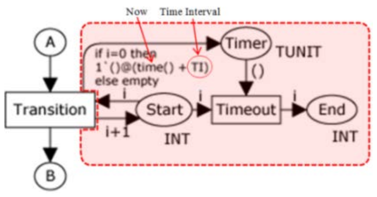

Figure 3. Transition with Time Interval

3.3 Transformation

When all BOMs are extended to E-BOMs with the proposed additional elements, our automatic transformation tool, transforms them into corresponding TCPN-component models (producing TCPN code for the three layers). The output is a .CPN file, with all the components generated as sub-modules. The modeler then composes all the components into a *TCPN-Composed Model* using CPN hierarchical tool. Then the model can be executed using CPN simulator and analyzed by performing state-space analysis.

3.4 State Space Analysis

In order to generate state-space of the entire model we use CPN state space calculation tool. When the state-space is generated, different query functions can be used to probe the state space graph for various verification questions. CPN tools provide some built-in-functions for the common query tasks. We additionally proposed functions to perform our model specific queries. In order to verify a composed CPN model, we propose a verification template that consists of the verification questions inform of three groups of properties:

A. General System Properties

State-Space analysis technique is very useful technique to verify general system properties such as *freedom of deadlock, livelock, starvation*, or *existence of boundedness, mutual exclusion, fairness, sequentiality, time-synchronization* etc. Choice of these properties as a verification criteria depends on the modeling objectives and their fulfillment become necessary conditions for the correctness of the composition. The solution for verifying a generic property involves specification of the property in CPN terms, and definition of a query function (or algorithm), to reason its satisfiability or violation e.g., freedom of deadlock property is specified in CPN terms as: "*An absence of a marking with no-out going arcs, in the entire state-space graph*". Its solution is provided inform of a built-in library function: `ListDeadMarking()` which returns a set of all those markings (if any) which have no outgoing arcs. If the result of this query is an empty list, then we assert that the model is deadlock free. Similarly CPN-tool has built-in solutions for other properties such as liveness, fairness and boundedness etc.

B. Goal Reachability

We propose to define the desired outcome of the composed model in form of a "Goal state", and use a "*Goal reachability*" function to assess if it is reachable in the state space or not. The goal-state can be viewed as a CPN based translation of the requirements specification. A typical goal state could be certain desirable values of state-variables in structural layer, reaching of particular state(s) in behavioral layer or producing some required data at output port(s) of the communication layer (or a combination of all the three), in one or more components of the composition. A composed model may have multiple goals.

C. Scenario Centric Properties

We also propose to define some safety (or unsafe) assumptions, which are particular to the scenario. They represent certain desirable (or un-desirable) situations which must (or must not) occur in order to satisfy (or violate) the requirements. These properties are not the ultimate goal(s), but they may become necessary conditions in order to reach the goals.

3.5 State space Query Functions

We develop a library of custom functions, using CPN-ML to perform verification of the properties, specified in the verification template. Some of these functions are explained as follows :

- **`GoalState()`**: Finding goal state reachability is not a standard operation, and depends on the way Goal-State is defined. Most commonly, we make use of our library functions: `IsEqual(), IsNot equal(), IsBetween(), IsUpperBound() or IsLowerBound()` to define a "predicate", that serves as a goal state reachability condition, and then use **`SearchNode()`**function to find those nodes, which satisfy the predicate. If one or more nodes are found, then it is verified that the goal is reachable. In cases, where it is important to know "how a goal is reachable", and which sequence(s) of the occurrence of transitions, lead to the goal(s), we use **`SearchArc()`**function with the predicate.
- **`minTime()/maxTime()/Interval():`** Finding nodes, having timed multi-sets with timestamps greater or lesser then a certain value, or between a certain time interval. These functions work on Timed CPN models.
- **`ExportGraph():`** We develop an export function, that creates a .DOT file of the entire state-space and can be viewed in graph tools such as GraphViz or Gephi, for visualization and performing further tests on the graph such as finding certain paths/shortest paths/longest paths between two particular nodes.

When, a composed model satisfies all the required system properties, qualifies its goal state reachability, and fulfills the scenario centric safety criteria, we say that it is verified at dynamic semantic composability level.

4 CASE STUDY: FIELD ARTILLERY

In (Mahmood, et al. 2012) we presented a case study of a Field Artillery model, to explain our verification process. In this paper we revise this case study to explain how time properties are defined and verified. This case study consist of two scenarios, based on two modes of artillery fire i.e., *direct fire* where the target is in the line-of-sight and *indirect fire*, where the target is out of sight, and artillery unit is requested for fire support by the forward observers. Both of these scenarios are supposed and designed to illustrate our verification approach of timed based systems.

4.1 Indirect fire scenario

In this scenario, following components are composed:

- Field component: Where enemy and friendly units are deployed.
- Observer: A soldier who observes enemy units at the forward location and coordinates fire support.
- BHQ: BHQ, supervises the entire operation of fire support at the battalion level.
- Battery: Three units of artillery batteries (cannons and crew) actually responsible to hit the target.

We assume that a soldier observes the field and detects enemy units. When a target is spotted, he calls BHQ for fire support and provides the target details. Time On Target (TOT) is the military co-ordination of artillery fire observed by multiple firing units, so that all the munitions arrive at the target at precisely the same time (or plus or minus three seconds from the prescribed time of impact). This is done in order to achieve maximum target destruction. BHQ assigns target to the batteries, and also schedules a certain "TOT" for the batteries to comply. Each battery needs some time to prepare, so that it can align itself for correct orientation and elevation by computing the target's range and bearing and load appropriate ammunition, we call it preparation delay (PD). Knowing the range of the target, and the muzzle velocity, it is possible to estimate the time of flight (TOF) i.e., the time between launch and impact of the round. Therefore the exact Time to fire (TTF) the round for each battery can be computed, as follows:

$$TTF = TOT - (\text{Time of target assignment} + PD + TOF) \quad (3)$$

Each battery fires at its TTF. When *field component* receives fire, and if the detonation is within a destruction radius, then the target is said to be destroyed otherwise it is missed. If all batteries mange to hit the target within TOT±3 we say, that a desirable property have been satisfied.

In order to proceed with our verification process, we assume that the BOM composition is given as an input. Figure 4 represents BOM state-machines of each component in the composition:

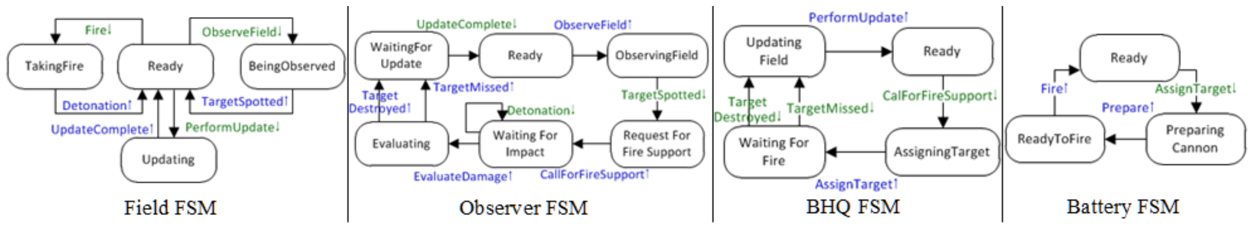

Figure 4. Field Artillery BOM State-machines

In figure 4, States are defined in rounded rectangles, whereas events are defined over the arrows. Blue color represents send events whereas green color represent receive events of state-machine.

Battery E-BOM						
States	Ready, PreparingCannon, ReadyToFire; Initial-State(s) = Ready					
State Variables	{ID, CT, TTF}					
Comm. Ports	IN-Ports = {AT}; Out-Ports={F}					
Transitions						
State	**Event**	**$\{SV_{In}\}$**	**$\{SV_{Out}\}$**	**Action**	**TimeFx**	**NextState**
Ready	AssignTarget	-	CT	-	-	Preparing
Preparing	Prepared	ID, CT	CT, TTF	act_1	PrepDelay()	ReadyToFire
ReadyToFire	Fire	CT ,TTF	-	act_2	TTF()	Ready
Notes: ID is the battery ID; CT stands for Current Target; The action functions act_1 & act_2 are defined using CPN-ML and can be viewed in the TCPN implementation; PrepDelay() is a random function using Normal(10,5) distribution; TTF() is a time-function defined using eq. 3; There are no guards.						

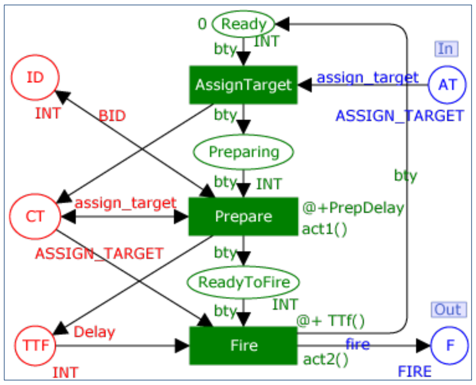

Figure 5. (a) Battery E-BOM (b) Battery CPN-Component Model

The scenario starts, when the Observer component sends *Observe-Field* from its Ready state, which is received by Field component (also at Ready state). We extend each BOM to E-BOM, and transform it to CPN component model using our transformation tool. Figure 5 presents the E-BOM and CPN Component Model representation of Battery component (as an example). For further details we refer the interested readers to download complete TCPN implementation of Field Artillery scenario from here.

Figure 5a describes the information that is extended in the BOM components including state-variables, communication ports and the extended transitions, whereas Figure 5b, illustrate structural layer (in red), behavioral layer (in green) and communication layer (in blue). When all components are transformed, they are composed in a TCPN composed model as shown in figure 6.

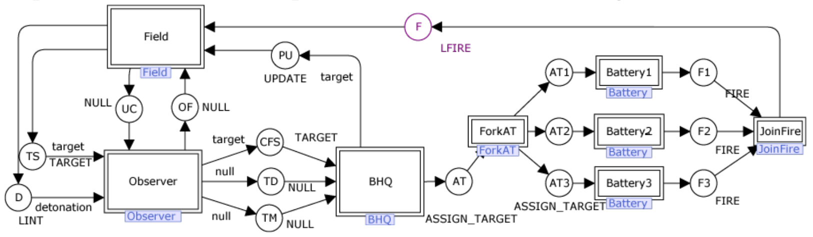

Figure 6. Field Artillery TCPN Composed Model

The circles represent socket places that connect components with each other. The rectangles represent components that are transformed form E-BOM (except two auxiliary components Join and Fork that facilitate composition of the components). The composed CPN model can now be executed and analyzed. We define the following verification template:

Property	**Definition**	**CPN Translation and verification method**
Goal State	All enemy units are destroyed	Search all nodes, where **Data** place of the **Field** components has an empty list
System properties	Deadlock freedom	There is no node that has no outgoing arc (except the goal states).
Scenario Centric properties	TOT as safety property.	Check if there is any node in the state space, where **F** place has three tokens (meaning all three batteries have delivered fire) and at least one of these tokens deviates from the TOT more than 3 time units, then the TOT property is violated. If no such node is retrieved, then TOT property is said to be satisfied. We apply minTime() and maxTime() functions to verify this property.

Table 1: Verification Template

For verification, state-space is calculated using CPN tools. We use our state-space analysis library functions to perform property verification according to our verification template. If all three properties are satisfied, we say that the model is verified at dynamic semantic level, and hence justifies the necessary condition of the overall composability.

In a counter example, we used a different field map, where enemy units are intentionally placed at a distance greater than the firing range of the batteries. So BHQ will assign these “unfeasible” targets to the batteries, which cannot be destroyed, hence the goal state will be unreachable. Similarly, we intentionally introduced an erroneous function, which causes the BHQ to compute TOT, so close to the preparation delay of the batteries, that in some replications, they batteries fail to satisfy TOT property, due to delayed launch, hence verification fails.

4.2 Direct fire scenario

In this scenario, we suppose that the batteries engage directly with the enemy at night, therefore in order to achieve advantage at night, illumination mortars are used to fire illumination rounds at the enemy location. They ignite and expose the exact position of the troops and vehicles. There is a time limitation for the ignition, so the batteries are supposed to hit maximum targets within in a given time interval. Based on this scenario, we replace the observer component with a mortar component in the composition as shown in figure 7. When *Fire-illuminator* transition occurs, a token is placed at the *Light* communication port and stays there (simulating the presence of light in the field) until the time interval expires. The illumination transition of the mortar component is bounded with a timer, to implement lapse of the assigned time interval (TI). The shaded green area represents timer implementation in CPN. TI is initialized by the state-variable *Interval*. According to the figure, the timer will stop when the model time reaches 100 time units from the time when illuminating transition is fired (for the first time). When TI is reached, the token is removed due to the firing of timeout transition. This component is connected with the batteries, which wait for the "illumination light" (at Light communication port), and shoot the targets, until there is light. Our goal state is same, i.e., *"all targets must be destroyed"*, but now batteries have a limited interval of time to reach the goal. We repeat the entire process, to verify that the composed model satisfy its requirements.

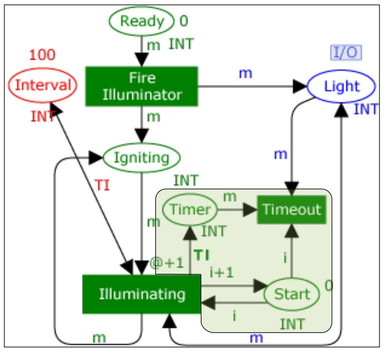

Figure 7. Mortar Component

This case study shows how time constraints can be introduced in a composed model, and verified using our verification approach.

5 SUMMARY & CONCLUSION

In this paper we present a process for the verification of composability at dynamic semantic level, with a focus on models of Real-time systems. We propose to use Base Object Model as a conceptual modeling framework, and Colored Petri Nets as an executable modeling standard. We suggest extending standard BOM model into E-BOM, so that it contains necessary behavioral details, required for its implementations, specially the time function. We provide an automatic transformation method to convert E-BOM into our proposed Timed CPN component model, which is useful to represent a model component in CPN language, yet it preserves the model structure and behavior conceptually intact. For the purpose of dynamic-semantic composability verification we suggest a verification template, as assessment criteria, which can be used to verify our executable model using State-space analysis. Lastly, we discuss a case study of Field Artillery, with two scenarios, and provide a counter example to show how our framework verifies a given composition, particularly at a dynamic semantic level.

Our proposed verification framework expedites the process of composability verification and provides suitable environment for finding out defects in the model composition. Moreover, the application of Colored Petri Nets and its analysis techniques in our approach is well justified and very constructive, due to its effectiveness in being able to model typical characteristics of dynamic real time event-driven systems, such as c*oncurrency*, *conflict*, and *sequential* execution. Moreover, because of numerous analysis methods and verification algorithms contributed by the CPN community over a couple of decades, CPN provides a significant improvement on efficient and accurate reasoning regarding the model correctness. We intend to fully automate the construction of TCPN composed model from the generated CPN components to further depreciate the manual human effort in the development.

REFERENCES

Balci, O, J D Arthur, and W F Ormsby. "Achieving reusability and composability with a simulation conceptual model." *Journal of Simulation* 5, no. 3 (August 2011): 157-165.

Balci, Osman. "VERIFICATION, VALIDATION AND ACCREDITATION OF SIMULATION MODELS." *Proceedings of the Winter Simulation Conference.* Atlanta, GA, 1997.

Christensen, Søren, Lars Kristensen, and Thomas Mailund. "A Sweep-Line Method for State Space Exploration." In *Tools and Algorithms for the Construction and Analysis of Systems*, by Tiziana Margaria and Wang Yi, 450-464. Springer Berlin / Heidelberg, 2001.

David, Alexandre , Oliver M Möller, and Wang Yi. *Verification of UML Statecharts with Real-Time Extensions.* Tehnical Report, Uppsala, Sweden: Department of Information Technology, Uppsala University, Wang .

Davis, Paul K., and Robert H. Anderson. *Improving the composability of department of defense models and simulations.* RAND National Defense Research Institute, 2003.

Elgaard, Louise . "The Symmetry Method for Coloured Petri Nets Theory, Tools and Practical Use." PhD Dissertation, Aarhus, Denmark, July 2002.

Fishwick, Paul A. *Simulation Model Design and Execution: Building Digital Worlds (1st edition).* NJ, USA: Prentice Hall PTR, 1995.

Gustavson, Paul . "Building and Using Base Object Models (BOMs) for Modeling and Simulation (M&S) focused Joint Training." *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)* . Orlando, Florida, 2005.

Hofmann, Marko . "Component based military simulation: lessons learned with ground combat simulation systems." *Proceedings 15th European Simulation Symposium.* Delft, Netherlands: SCS European Council , 2003.

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

Jensen, Kurt, Søren Christense, and Lars M Kristensen. "CPN Tools State Space Manual." Manual, Aarhus , Denmark, 2006.

Kristensen, Lars Michael . "State Space Methods for Coloured Petri Nets." Ph.D. Dissertation, Department of Computer Science, University of Aarhus, Aarhus, Denmark, 2000.

Mahmood, Imran, Rassul Ayani, Vladimir Vlassov, and Farshad Moradi. "Verifying Dynamic Semantic Composability of BOM-based composed models using Colored Petri Nets." *To appear in: 26th Workshop on Principles of Advanced and Distributed Simulation.* Zhangjiajie, China, 2012.

Moradi, Farshad, Rassul Ayani, Shahab Mokarizadeh, Gholam Hossein Akbari Shahmirzadi, and Gary Tan. "A Rule-based Approach to Syntactic and Semantic Composition of BOMs." *11th IEEE Symposium on Distributed Simulation and Real-Time Applications.* Chania, 2007.

Olderog, Ernst-Rudiger, and Henning Dierks. *Real-time systems - formal specification and automatic verification.* Oldenburg, Germany: Cambridge University Press, 2008.

Petty, Mikel D. "Verification and Validation." In *Principles of Modeling and Simulation*, 121-149. John Wiley & Sons, Inc., 2008.

Petty, Mikel D., and Eric W. Weisel. "A theory of simulation composability." Virginia Modeling Analysis & Simulation Center, Old Dominion University, Norfolk, Virginia, 2004.

Robinson, Stewart , Roger Brooks, Kathy Kotiadis , and Durk-Jouke Van Der Zee. *Conceptual Modeling for Discrete-Event Simulation.* FL, USA: CRC Press, Inc., Boca Raton, 2010.

Saadawi , Hesham , and Gabriel Wainer. "Verification of real-time DEVS models." *Proceedings of the Spring Simulation Multiconference.* San Diego, CA, USA, 2009 .

SISO-I. *Base Object Model (BOM) Template Specification.* Simulation Interoperability Standard Organization (SISO), Orlando, FL USA: SISO Base Object Model Product Development Group, 2006.

Szabo, Claudia , Yong Meng Teo, and Simon See. "A Time-based Formalism for the Validation of Semantic Composability." *Winter Simulation Conference.* TX, USA, 2009. 1411-1422.

Tolk, Andreas. "Interoperability and Composability." Chap. 12 in *MODELING AND SIMULATION FUNDAMENTALS Theoretical Underpinnings and Practical Domains*, by John A. Sokolowski and Catherine M. Banks. John Wiley, 2010.

Westergaard, Michael , Lars Michael Kristensen, Gerth Stølting Brodal, and Lars Arge. "The ComBack Method – Extending Hash Compaction with Backtracking." In *Petri Nets and Other Models of Concurrency,* by Jetty Kleijn and Alex Yakovlev, 445-464. Springer Berlin / Heidelberg, 2007.