Verifying Dynamic Semantic Composability of BOM-Based Composed Models Using Colored Petri Nets

Conference Paper · July 2012

DOI: 10.1109/PADS.2012.48

CITATIONS
8

READS
133

4 authors:

Imran Mahmood
KTH Royal Institute of Technology
16 PUBLICATIONS **42** CITATIONS
SEE PROFILE

Rassul Ayani
KTH Royal Institute of Technology
83 PUBLICATIONS **1,042** CITATIONS
SEE PROFILE

Vladimir Vlassov
KTH Royal Institute of Technology
122 PUBLICATIONS **772** CITATIONS
SEE PROFILE

Farshad Moradi
Swedish Defence Research Agency
37 PUBLICATIONS **343** CITATIONS
SEE PROFILE

2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation

Verifying Dynamic Semantic Composability of BOM-based Composed Models using Colored Petri Nets

Imran Mahmood, Rassul Ayani, Vladimir Vlassov
KTH Royal Institute of Technology
Stockholm, Sweden
{imahmood, ayani, vladv}@kth.se

Farshad Moradi
Swedish Defense Research Agency (FOI)
Stockholm, Sweden
farshad@foi.se

***Abstract*—Model reuse is a promising and appealing convention for effective development of simulation systems as it offers reduction in development cost and time. Various methodological advances in this area have given rise to the development of different component reusability frameworks such as BOM (Base Object Model). But lack of component matching and weak support for composability verification and validation, in these frameworks, makes it difficult to achieve effective and meaningful reuse. In this paper we focus on Composability verification and propose a process to verify BOM based composed model at dynamic semantic level. We suggest an extension to the BOM components, to capture behavior at a greater detail. Then we transform the extended BOM into our proposed Colored Petri Nets (CPN) based component model so that the components can be composed and executed at an abstract level. Subsequently we advocate to use CPN tools and analysis techniques to verify that the model satisfy given requirements. We classify the properties of a system among different groups and express the model's requirements by selecting some of the properties from these groups to form requirement specification. Also we present an example of a Field Artillery model, in which we select a set of properties as requirement specification, and explain how CPN state-space analysis technique is used to verify the required properties. Our experience confirms that CPN tools provide strong support for verification of composed models.**

Keywords—Model Verification; Model Transformation; Dynamic Semantic Composability; Colored Petri Nets; State-space Analysis; Deadlock; Field Artillery scenario.

I. INTRODUCTION

With the advent of net-centric era of methods and technologies in designing complex simulation systems, the focus of Modeling and Simulation (M&S) industry has been driven by the most recognized potential benefits: reduced development cost and time [1]. The community has also taken deep interest in the quality design principles and their underlying supportive theories that promise these benefits, Most important of these are reusability and composability [1] [2]. Reusability is the ability of simulation components to be reused for different applications. Composability is the capability to select and assemble components in various combinations to satisfy specific user requirements [3]. Composability is one of the effective means to achieve reusability as it offers a reduction in the complexity of system construction by enabling the designer to reuse appropriate components without having to re-invent them [4]. But it is a challenging and daunting problem in practice, and is considered to be the elusive holy grail of modeling and simulation [5].

Composability contends with the alignment of issues on the modeling level [6], therefore at an abstract model level, it is the creation of a complex model from a collection of basic reusable model components. The term Composability carries various meanings and views in theory which are distinguished, primarily by its different "levels" or "layers", as introduced by different researchers. It is essential to contemplate these different composability "levels" to better understand its theoretical underpinnings. Tolk [7] propose a six layered model of component composability, namely *technical*, *syntactic*, *semantic*, *pragmatic*, *dynamic*, and *conceptual layer*. Similarly Medjahed et al [8] introduce a *composability stack* in which the composability of semantic web services is checked at four levels: *Syntactic*, *Static Semantic*, *Dynamic Semantic* and *Qualitative level*. In M&S three of these levels were brought into consideration by Farshad et al [9] in the composition process of the model components. *Syntactic composability* means that components can be connected to each other and they can operate together. *Static-Semantic composability* refers to the fact that the coupling of components is considered meaningful and they have the same understanding of concepts to communicate with each other. *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 M&S, verification is typically defined as a process of determining whether the model has been implemented correctly [10]. 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 [11]. We concentrate on the former part and assume that the behavioral correctness is an integral part of the model's requirements.

In this paper, our focus is the Dynamic Semantic level of composability, where we aim to verify the behavioral correctness of composed components. We define dynamic semantic composability verification (or simply behavioral verification) as a process of determining whether a composed model satisfies given requirements, where the requirement specification consists of system properties such as *deadlock freedom, livelock freedom, mutual exclusion, and fairness*. These properties are classical cases of system behavior and their satisfaction may be required for the behavioral correctness of the composition e.g., the absence of a deadlock implies that the components will never halt infinitely waiting for each other. The selection of these properties as a requirement depends on the goals of the composition. System properties may also include scenario specific properties representing certain desirable or undesirable incidences in the system. In behavioral verification of a given composition, we analyze that the model components are correctly selected and assembled such that they satisfy their requirement specification and their combined behavior is suitable to reach their composition goals.

Composability essentially relies on a suitable composition framework that can provide accurate reasoning of correctness at each level. Base Object Model (BOM) is a SISO (Simulation Interoperability Standards Organization) certified component

978-0-7695-4714-5/12 $26.00 © 2012 IEEE
DOI 10.1109/PADS.2012.48

architecture that is partially consistent with this stipulation [12]. It contributes to conceptual modeling by providing the needed formalism and influences the ability to develop and compose model components [13]. BOM development effort was initiated to serve as a standard description of simulation components for HLA (High Level Architecture), which is IEEE certified architecture for distributed simulations. In BOM different elements such as entities, events, actions and state-machines of the components are defined. Entities, events and actions represent the structural information about the real world objects that are being modeled, whereas state-machine formalizes component behavior. For details interested readers should refer to [14]. 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 system behavior and reasoning about the validity of the composability at dynamic semantic level [9]. This fact leads us to the investigation of external methods for the specification of additional modalities that are involved in the behavioral modeling of components and methods for their composability verification.

Different approaches have been suggested for the external composability verification. A rule-based approach proposes to match a set of BOMs and verify their syntactic, static-semantic and dynamic-semantic composability [9]. A similar work [15] proposes an instrumentation technique for specification of system properties and verification of behavioral composability using a *Model Tester*. Another method [16] supports the idea of using Petri Nets algebraic techniques for the composability verification, where fairness is considered as a behavioral property. A different approach addressed the semantic validation of composed models in [17].

In this paper, we present an approach to verify composability of BOM components at dynamic semantic level. At first, we suggest to apply the concept of "Extended finite state-machine" (EFSM) given in [18] as a formal specification and propose to extend BOM's conceptual modeling segment, in order to cover its existing deficiencies of the modalities. We refer to our extended version of BOM as **Extended BOM or E-BOM)** throughout the paper. It should be noted that E-BOM is not a SISO standard. Secondly, we present a method to automatically transform each extended BOM (E-BOM) component in to our proposed CPN based component model. This component model represents all the essential parts of a model (like BOM) in form of a CP-Net. It is called component model because it is generated as a CPN module and can be composed and executed in CPN environment [19]. Consequently, we utilize numerous analysis and verification techniques (mainly contributed by CPN community) for evaluating composability at behavioral level. In essence, given a BOM composition and the requirements specification, we transform each E-BOM into CPN component model; compose them as a hierarchal CP-Net, and apply verification techniques such as state-space analysis to evaluate the required system property(s) and show that the model fulfills given requirements. Our approach varies from previously specified related approaches (and with others in general) in a sense that we utilize the expressive capability of CPN framework in the modeling of complex systems involving concurrency, synchronization and communication, using component based approach. Our approach is also favored by the formal semantics of CPN and the abundance of CPN verification techniques, contributed by the CPN community over two of decades. In our observation, using CPN for verification proves to be more fruitful as compared to other similar formalisms, due to the broader range of verification methods and techniques and due to their greater suitability in formal representation and evaluation of concurrent behaviors.

The rest of the paper is organized as follows: Section II briefly defines and explains basic concepts and formalisms used in this paper. Section III formulates our methods and approach for extending BOM to E-BOM, the transformation of E-BOM models to CPN component models, their composition, and the verification of the composed model using the methods available in CPN environment. Section-IV furnishes implementation details of our verification process and the proposed framework. In Section V we discuss a case study of a Field Artillery model to explain our approach whereas section VI frames the summary and conclusion.

II. Basic Concepts and Formalism

In this section we briefly discuss basic concepts of Extended Finite State-machine, Colored Petri Nets and some important concepts used in our work. We also define requirement specification and classify families of verification properties in this section.

A. Extended Finite State-machine

An Extended Finite State Machine (EFSM) is defined by the tuple

$$\mathbf{M = (Q, I, \Sigma_1, \Sigma_2, V, \Lambda)}$$

Where:

- Q ($\neq\emptyset$) is a finite set of states.
- $I \subset Q$ is the set of initial states
- Σ_1 is a finite set of (send or receive) events.
- Σ_2 is a finite set of actions
  (Where *actions* are the instructions to be executed and should not be confused with the BOM actions, which are used in pattern of interplay).
- V is the set of state variables.
- $\mathbf{\Lambda}$ is a set of transitions; each transition $\boldsymbol{\lambda \in \Lambda}$

$$\lambda = q \xrightarrow{\mathbf{e\ [g]\ /\ a}} q'$$

Where

- q and $q' \in Q$
- $e \in \Sigma_1$ is an event
- g is a condition (or guard)
- $a \in \Sigma_2$ is an action.

It means if the system is at a state 'q', and an event 'e' occurs, and if the guard 'g' is satisfied, then 'a' action is executed, and the system will transit to the next state q′. The motivation behind using EFSM has following arguments:

- By introducing state-variables in FSM, we are able to model the structural attributes of a component, which may be affected due to the change of states and occurrence of transitions (behavior), and can help in making the model more realistic. Also the values of these attributes can be used in the arithmetic or logical evaluation to assess trigger conditions of the transitions. It can also be useful to transfer variable values from one model to another model, thus in a sequential composition, a value output by one component can be consumed by the other in the composition [18].
- By introducing actions, during an occurrence of a transition, we can capture complex behavior that cannot be obtained through simple transition labels (such as events in standard BOM), because actions (set of instructions) can cause changes in the state-variables making the behavior of the model more realistic [18].

We apply the concept of EFSM to the BOM conceptual model, so that we can incorporate state-variables and extended representation for transitions (events, guards, actions), to a form, which we name: Extended BOM or E-BOM.

B. Colored Petri Nets

Coloured Petri Nets is a graphical language for constructing models of concurrent systems and analyzing their properties. CPN is a general purpose discrete event language, combining the capabilities of Petri nets, as a foundation of the graphical notation and the basic primitives for modeling concurrency, communication, and synchronization, and a programming language (CPN ML), which is based on Standard ML functional programming language, that provides the primitives for the definition of data types and for specifying data manipulation routines [19]. CPN is formally defined by the tuple:

CPN = (P, T, A, Σ, V, C, G, E, I) where:

- P is a finite set of places
- T is a finite set of transitions such that: $P \cap T = \varnothing$
- $A \subseteq P \times T \cup T \times P$ is a set of directed arcs.
- Σ is a finite set of non-empty colour sets.
- V is a finite set of typed variables such that:
- $Type[v] \in \Sigma$ for all variables $v \in V$
- C: $P \rightarrow \Sigma$ is a colour set function that assigns a colour set to each place.
- G: $T \rightarrow$ Expression is a guard function that assigns a guard to each transition t
- E: $A \rightarrow$ Expression is an arc expression function that assigns an arc expression to each arc a
- I: $P \rightarrow$ Expression is an initialization function that assigns an initialization expression to each place p.

Detailed explanation of the concepts required to work with CPN is beyond the scope of this paper hence interested readers are recommended to consult: [19] [20] [21]. "CPN Tools" is a software package for the editing, simulation, state space analysis, and performance analysis of CPN models [21]. The tool acts like an integrated development environment (IDE) for the construction of CPN models. It comes along with a bundled simulator that efficiently handles the execution of untimed and timed nets. The most important feature of CPN tool from our point of view is the generation and analysis of state spaces. The analysis of state space includes various built-in state-space querying functions, and support for creating analysis report which altogether greatly contributes to the verification process.

C. Hierarchical Coloured Petri Nets

CPN model can be organized as a set of modules; where modules can be seen as black boxes which make it possible to work at different abstraction levels, concentrating on one at a time. 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. This way, the entire model can be divided into simpler modules (or components), and can be composed in a separate model to promote modular development and flexible reuse [19] [21]. Our CPN component model uses CPN hierarchical features.

D. State Space Analysis

State space method is one of the most prominent approach for conducting formal analysis and verification. The basic idea is to calculate all reachable states and state changes of the system and represent these as a directed graph. From a constructed state space it is possible to 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 a good state, and never reach a bad state and the guarantee of reaching a goal state(s).

E. Requirement Specification

For the purpose of verification, we assume that the given composed model under consideration is accompanied with a set of requirement specifications, consisting of a set of properties. These properties are typically classified in to following four families (or kinds) of properties: [22]

- ***Reachability Properties*:** express that some particular situation (desirable or undesirable) can be reached. (e.g., goal reachability)
- ***Safety Properties*:** suggests that under certain conditions, something bad will never occur (e.g., deadlock, nuclear meltdown)
- ***Liveness Properties*:** express that under certain conditions, something good will eventually occur. (e.g., a component will eventually enter critical section).
- ***Fairness Properties:*** state that under certain conditions, something good will occur infinitely often (e.g., fair distribution of shared resource)

We assume that based on the modeling objectives, some of these properties are formulated in the requirement specification. We assert that the components are composable at the dynamic semantic level, if they fulfill their requirements.

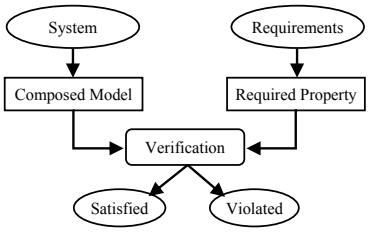

Figure 1. Composition & Verification Relationship

III. CPN BASESD COMPOSABILITY VERIFICATION

This sections presents methods and approach as our contribution, realized at different stages of component based modeling and simulation development life-cycle.

The key milestones of this lifecycle are described in fig 2. A simuland is the real world system of interest, which is to be simulated [11]. Based on the abstractions that the modeler has about a simuland and the requirements in terms of goals and objectives, the concerning model components (such as BOM), are discovered from a component repository and composed, to form a conceptual model. The conceptual model is implemented in form of an executable simulation (such as HLA) which is executed and results are generated that can be used for refinements.

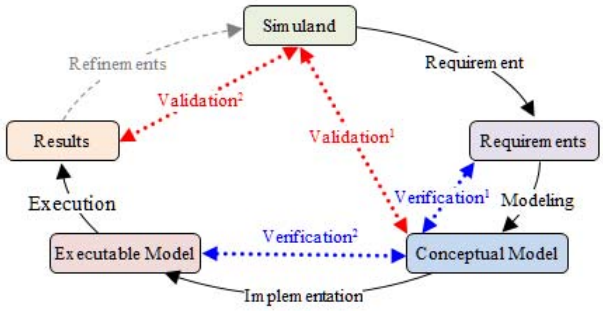

Figure 2. Development Life-cycle (inspired from [11])

In this paper, our focus is centered on "Verification[1]" where we examine that the conceptual model correctly represents the given requirements, (some people call it validation). For this purpose we transform our conceptual model (BOM composition) into a CPN model and apply various verification techniques. We propose this transformation process in three steps:

A. BOM to E-BOM

In this step, each BOM component of the composition, is parsed and is presented to the modeler for applying additional information, such as state variables data-types, guards and actions (which are not part of the standard BOM). This is important to note that we don't intend to modify the structure of the standard BOM; instead we extend it using EFSM formalism to serve our purpose. Figure 3 describes the mapping from BOM to E-BOM, showing all the elements that can be imported from BOM model and the rest that modeler needs to specify explicitly.

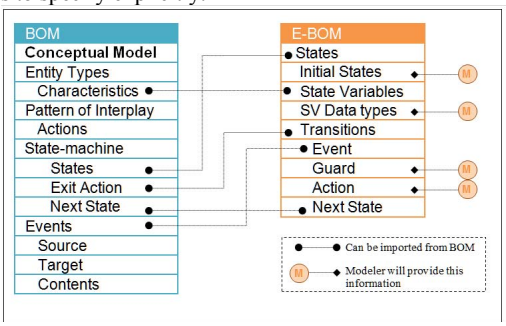

Figure 3. BOM to E-BOM extension

B. E-BOM to CPN Component Model

At this step, we transform each E-BOM to our proposed CPN component model. It is basically a three layered representation of a component in form of CP-Net:

1) Structural Layer

This layer consists of the physical attributes of the CPN component model, and are represented inform of places. Each state-variable in the E-BOM, becomes a place, and its data type becomes a *Color Set* of that place. If a state-variable has a primitive data type such as: INT, STRING or BOOL, then we simply assign it to the place, but if the data type is complex, such as *product, tuple, record* or *list*, then we first declare that data type as a *Color Set* and then assign it to the place. The places are also initialized with their initial value, if they are marked as initial states in E-BOM.

2) Behavuoural Layer

This layer represents the behavior of CPN component model. The state-machine in E-BOM becomes a part of this layer, such that each state in the E-BOM becomes a place of type INT, and the initial state(s) is initialized with a zero value token. And each transition in the E-BOM becomes a CPN transition in this layer, and so are guard and action, which are translated into guard and code-segment inscription respectively [21] (Section: Documentation\Concepts\Inscriptions & expressions\Transition inscriptions). The purpose of this layer is to be able to represent state-machine behavior of a component in form of CP-Net, where the flow of token(s) represents the change of component's state. The transitions of this layer are connected to those state variables (places) of the structural layer, where an update (read or write operation) is necessary. If a transition reads a variable value, then it is connected with an incoming arc, whereas if it writes a variable value, then it is connected with an outgoing arc. For this matter, we also propose a "Data Marshaling" technique using CPN ML functions, so that the mismatch of Color Sets between the layers can be accommodated. This mismatch may occur, where the input and output places have complex data types of varied tuple size. In short, this technique can be perceived as Type casting of the color sets, and is necessary for the CP-Net to fulfill the CPN syntax.

3) Communication Layer

This layer is responsible for the interaction and communication among the components involved in the composition. It provides interface for connecting the inputs and outputs of the components and also provide information about the data exchange. In this layer, we create port places [21] which are connected to each transition of the behavioral layer. Note that if a transition represents a send event (of BOM), then it will be connected to an out-port place, with an outgoing arc, whereas if it represents a receive event, then it will be connected to an in-port place with an incoming arc. The type of the port place depends on the parameters (contents of events in BOM) of the transition, based on which we construct a color set of type *product*, and assign this color set to the port place. All these port places in each CPN component are connected to their respective "*socket places*" [21] in the main CPN model, where all sub components are composed. Figure 4 gives an overview of this transformation using a simple example. Figure 4a shows BOM FSM of a component, which is extended to E-BOM shown in Figure 4b.

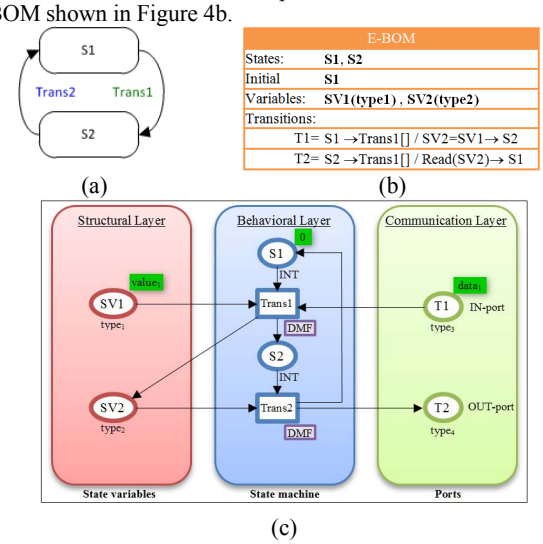

Figure 4. (a) BOM FSM (b) E-BOM (c) CPN component Model

In the E-BOM, two variables *SV1* of *type1* and *SV2* of *type2* are added. Also two transitions *T1* and *T2* are defined. *T1* has an event *Trans1* and an action: *SV2=SV1* whereas *T2* has an event *Trans2* and an action: *Read (SV2)*. Figure 4c represents CPN component model of this example. It is apparent from figure 4c that *Trans1*, in the behavioral layer can only be enabled if the place *SV1* is initialized with an initial value token (represented by a shadowed box), place *S1* has a token with zero value, and the place *T1* receives "$data_1$" token from some other component, which carries event parameters. When *Trans1* is fired, it consumes "$data_1$" token from *T1* port, reads value from variable *SV1;* it may use data marshaling function (DMF) to type cast color-sets; copies *SV1* to *SV2,* and changes component's current state from *S1* to *S2*. Similarly, in the next step *Trans2* reads from *SV2,* and sends its value as a token at port *T2,* before it returns to state *S1*. Intuitively, the role and nature of these three layers are different but they work together

to make the CPN module act like a generic executable model component.

In order to automate the E-BOM to CPN transformation, we develop a CPN-XML writer application, which takes E-BOM component information as input and produces CPN-XML code for all three layers of CPN component model, of the entire BOM composition. For each component, a separate CPN sub-page is generated (programmatically) and the necessary CPN elements (places, transitions, arcs, color sets, variable declarations, initial markings multi-sets, guards, actions, code segments, CPN ML functions, ports, ports-tag) are generated in one CPN output file, which can be loaded in CPN tools. At this moment, we propose to create a main model and manually combine the generated modules (using CPN hierarchical features). The output of this step is a composed CPN model.

C. Verification of Composed CPN Model

In this step, we perform the verification of the composed CPN model using State space analysis which involves three steps:

1) State space calculation

In step1, we perform standard procedure to generate state-space of the entire model using CPN state space calculation tool.

2) State space Query

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 propose additional functions to perform model specific queries. In order to verify a composed CPN model, we propose a verification template that consists of the verification questions in form of three groups of properties:

a) General System Properties:

There are some built-in and useful functions to verify general system properties such as deadlock freeness, liveness, fairness. 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 any node in the state-space graph, which has no outgoing arcs". CPN tool has built-in solutions for verification of some of the general system properties such as deadlock, liveness, fairness and boundedness.

b) Goal Reachability

We propose to create a "Goal reachability" function. Modeler can define the desired outcome of the composed model in form of a "Goal state" and try to evaluate whether the goal is reachable in the state space. 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 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 as well. In our verification framework, we include State-Space search functions to perform all these types of goal reachability.

c) Scenario Specific Properties:

The modeler may define certain safety (or unsafe) assumptions according to the context of the scenario. Essentially, these kinds of properties are not the ultimate goal, but there are certain desirable (or un-desirable) activities which must (or must not) occur in order to satisfy the requirements. These properties can be defined and verified in terms of reachable (or unreachable) markings, of the state space. We discuss this step with an example in our case study section.

3) Reporting/Visualization

Step 3 is an optional but useful step from the verification point of view. Here we create a report in text format (CPN tools built-in feature). We also provide a function to export state-space graph in *dot file format* which can be imported in any graphing tool (such as GraphViz, Gephi), for visualization purpose. Using the graph many observations, such as shortest path from initial marking to the goal state (marking), paths leading to deadlock, or cycles of livelock can be observed, and used to study the model for refinements.

IV. Verification Framework

We propose a verification framework, which consists of tools to perform all the steps mentioned above. Figure 5, describes an overview of our verification framework, where numbered red arrows indicate the sequence of the process flow. At first, BOM parser takes a set of BOM components, as input and passes the information to the E-BOM editor. This utility is used to map existing BOM elements to E-BOM and also to take input from the modeler, as described in figure 3. The output of this utility is a set of E-BOMs, which are subjected to E-BOM-to-CPN transformer at step 4. This procedure invokes CPN-XML writer and constructs corresponding CPN modules, based on our proposed CPN component model. The output of this procedure is a ".cpn" file, which can be opened and viewed using CPN tools. In the next step (6) all CPN modules are combined to construct a CPN based composed model, which is ready for both simulation and verification. Simulation can be performed using CPN simulation tool, which supports design of experiment (using different initialization settings) and collection of results. For verification we use the CPN-tools to perform state-space analysis.

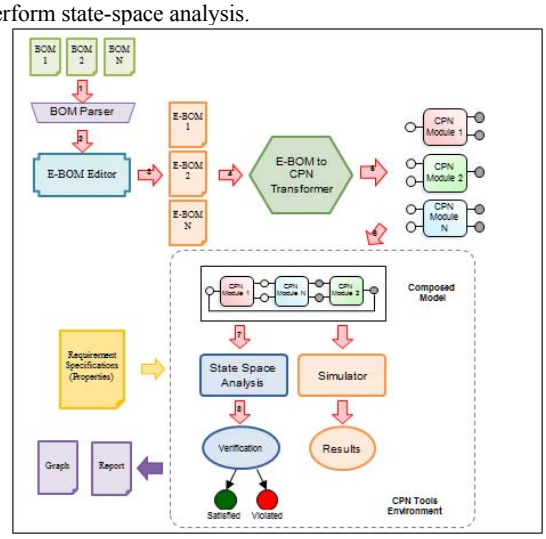

Figure 5. Verification Framework

Once the verification step is performed and if the model satisfies required properties (given in the requirement specification) we say that the model is composable at dynamic semantic level, which is a necessary condition for the correctness of the overall BOM composability.

V. Case Study

In this section we discuss a case study of a Field Artillery model. This model is composed of the following components shown in Fig 6:

- Field component: Where enemy and friendly units are deployed, but also some neutral objects are present.

With a given set of initial states, this component is used to model battle field.

- Observer: A soldier who observes enemy units at the forward location; initiates and coordinates fire support.
- Field Artillery: This is a composed model of a Field Artillery Battalion and has following sub-components:
  - BHQ: Battalion head quarter, supervises the entire operation of fire support at the battalion level.
    FDC: Fire direction Centre, performs tactical and technical fire direction. Validates target assignments in tactical terms (i.e.,. target is an enemy, not a neutral or friendly unit, checks target priority) and in technical terms (target is in range, unit have enough resources and appropriate ammunition to hit the target etc.)
  - 3x Batteries: Three units of batteries (cannons and crew) actually responsible to hit the target, based on the technical information: orientation, elevation, range (target distance) etc.

For the sake of simplicity and due to lack of space, we have reduced certain details which are present in actual indirect fire procedures. Figure 6 represents components of Field Artillery Model and the interactions between each component.

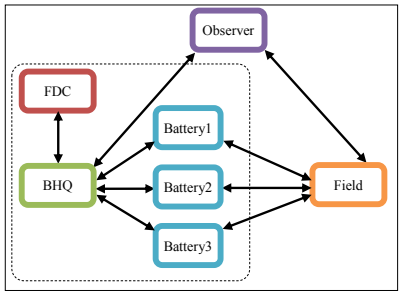

Figure 6. Field Artillery Component Model

We consider an Indirect Fire Support scenario, where the enemy units are not in the line of sight of the firing units. A soldier observes the field and detects enemy units. When a target (enemy unit) is spotted, he calls BHQ for fire support and provides the target details. BHQ requests FDC to process the target (tactically & technically). If the target is valid FDC approves the request otherwise the request is denied. If the request is approved BHQ assigns the target to the batteries. We suppose that the target can be one of three types: light (camps, troops, trucks), medium (tanks, light guns) or heavy (artillery units, missile launchers); and is assigned to one, two or three batteries respectively. This is because medium and heavy targets require the fire power of more than one battery for complete destruction. Based on this assumption, BHQ assigns target to the batteries. Battery components align themselves for correct orientation and elevation by computing the target's range and bearing (angle), load appropriate ammunition and fire the round. When Field components receives fire, and if the detonation is within a destruction radius then the target is said to be destroyed otherwise it is missed. This information is sent to the observer, who relays it to BHQ (indirect fire procedure). In a more detailed implementation of this model, we also model the procedures of target adjustments, where observer provides target adjustment data (if it is missed) and the firing procedure is repeated until the batteries are exactly aligned to hit the target, then fire for effect (FFE) takes place. Figures 7-11 describe BOM state-machines of each component.

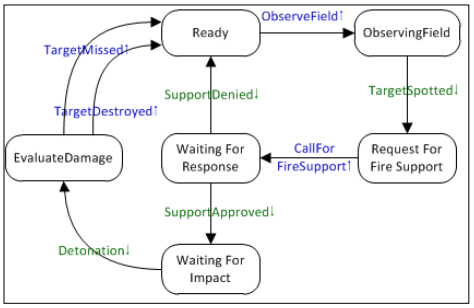

Figure 7. State machine of the Observer

[↑=SendEvent] [↓=Receive Event]

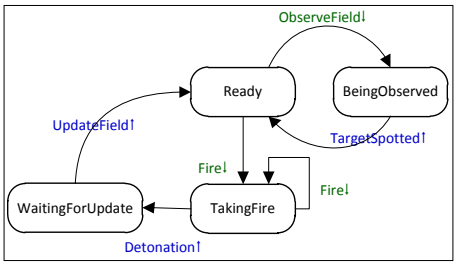

Figure 8. State machine of the Field

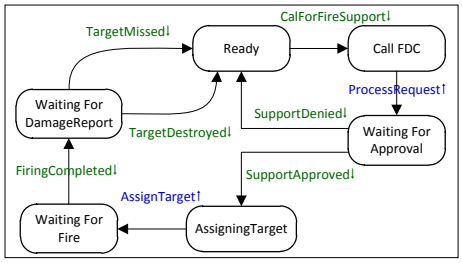

Figure 9. State machine of the BHQ

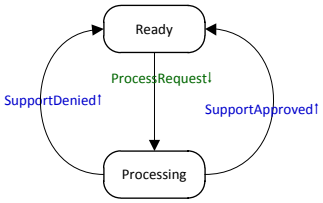

Figure 10. State machine of the FDC

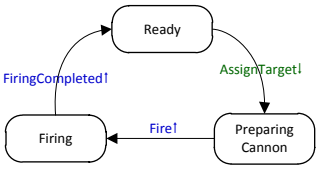

Figure 11. State machine of the Battery

Each component is parsed and extended to E-BOM. (E-BOM of FDC component is defined in Fig. 12 as an example). States Q and Events Σ1 are inherited from BOM (see Fig 10). We set *Ready* as an initial state. Also we define three variables. Field_Data stores information about the objects in the field. **Current_Target** stores the ID of current target under processing. **Result** stores a Boolean result of the processing. We also define transitions λ_1, λ_2 and λ_3. λ_1 takes **ProcessRequest** as an input (receive event) and executes

action $\mathbf{a_1}$ unconditionally (no guard). $\mathbf{a_1}$ is defined in CPN-ML notation which cross checks field data for the type of current target. If it is a valid enemy target then the **Result** is true else false. When this action is executed, Result variable is overwritten. Transitions $\boldsymbol{\lambda_2}$ and $\boldsymbol{\lambda_3}$ are concurrent transitions and are guarded by a shared condition; only one of them will be fired based on the value of **Result** variable.

Q = {Ready, Processing}
I = {Ready}
Σ_1 = {ProcessRequest, SupportApproved, SupportDenied}
Σ_2 = {a_1}
V = {Field_Data:record, CurrentTarget:int, Result:Boolean}
λ_1 = Ready ➔ ProcessRequest [ ] / a_1 ➔Processing
λ_2 = Processing➔ SupportApproved [Result=True] ➔Ready
λ_3 = Processing ➔SupportDenied [Result=False] ➔ Ready

Figure 12. FDC E-BOM

Similarly we extend each component to E-BOM, which is further transformed to CPN module in the next step in such a way that all variables are added in the Structural Layer and the State-machine is transformed into the Behavioral Layer. In communication layer, receive-events are transformed into input ports and send-events are converted into output ports. Figure 13 represents the CPN component model of the FDC.

Similarly all E-BOMs are transformed into CPN modules in the same fashion. (Note: details and complete implementation of this case study can be viewed at: (http://web.it.kth.se/~imahmood/FieldArtillery). In the next step all CPN modules are combined together through socket places in a CPN Composed Model as shown in figure 14. We have also introduced general purpose modules such as Join and Fork to facilitate the composition.

In the next step we generate state space of the entire Field Artillery Model using CPN state-space calculation tool, and perform verification. The generated state-space graph consists of 1970 nodes and 6486 arcs. (Figure of the state-space graph can be located in the link specified above).

In this particular scenario, we assume the following properties as our requirement specification:

- *Deadlock freedom*
- Goal-state reachability (i.e., *all enemy targets are destroyed*.
- *Avoidance of friendly fire* (scenario specific property).

Based on these assumptions, we verify the composed model as follows:

A. General System Property: (Deadlock freedom)

After the state-space is created, we execute a built-in library function: **ListDeadMarkings(),** that searches the entire state-space graph for dead-markings (i.e., the nodes which don't have outgoing arcs). If the result is an empty set, then the model is said to be deadlock free, otherwise this function will return all the dead-markings present in the composed model. At present, no such marking was detected.

B. Goal Reachability Property: (All enemies are destroyed)

The field component consists of a **Field_Data** place (as state-variable) which consists of tokens representing enemy units present in the field. When an enemy target is destroyed, the action of **UpdateField** transition (fig. 8) is responsible to update this variable, by eliminating the entry of the enemy object. When all enemy objects are eliminated, our goal is said to be reached. This can be verified if there exists a marking in the state space, in which **Field_Data** carries an empty list. For searching such a marking, we create a predicate that acts as a search criteria in the **SearchAllNodes()** function**.** When this function is executed, it finds all nodes in the state-space, where the predicate function evaluates the condition:

Mark.Field'Field_Data = []

If such node(s) is found then we can assert that the goal state is reachable. In this scenario, we observe that more than one such marking were found, because the model executes in a circular loop.

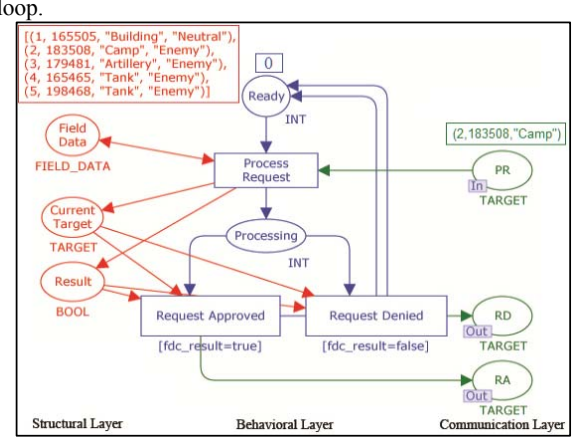

Figure 13. FDC component model

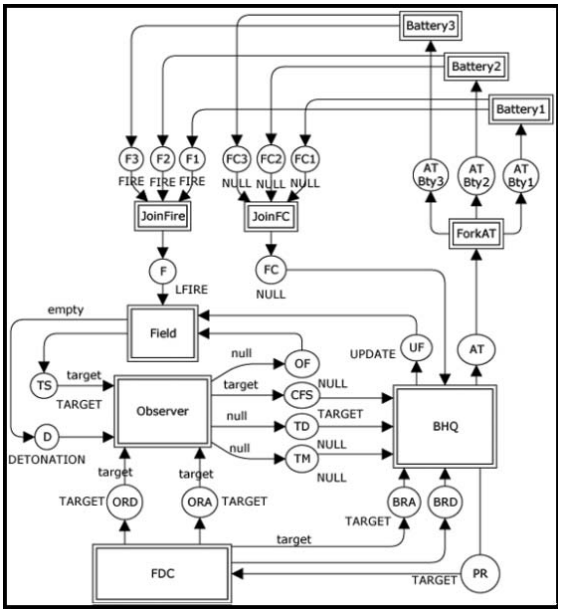

Figure 14. Field Artillery Composed Model

C. Safety Property: (Avoidance of friendly fire)

Similarly, to verify that the friendly fire never occurs in our model we search for the arcs: ***TargetDestroyed*** in the state space, where **Target_id** of the object is not an enemy (i.e., either it is friendly or neutral). The absence of such arc(s) will indicate that a friendly fire never occurs. We use **SearchAllArcs()** function to perform this property verification.

In the counter example of this scenario, we select a different BHQ component that asks for security credentials of the observer when he requests for fire support. Since this behavior is not modeled in our existing observer, there will be a deadlock (as Observer will expect BHQ to process the call for fire support, whereas BHQ will expect observer to reply to the security credential question).

Also we introduce the factors of target distance and firing range in our model that causes violation of goal state

reachability, if the assigned targets are beyond the firing range of the batteries. Because batteries will try to hit the assigned target, but due to firing range limitation, the round will never enter the destruction radius, and hence the goal state will never be reached (and the same target will keep getting assigned by BHQ). Similarly, we introduce an erroneous FDC component in the composition approves fire support without the discrimination of enemy or friendly units and causes a violation of "Friendly fire avoidance" property because such ***TargetDestroyed*** *arcs* are detected where the **Target_id** is a friendly or a neutral unit. Hence, using state space analysis, we can verify given requirements and if all requirements are satisfied we say that the composition is valid at the dynamic semantic level.

VI. Summary and Conclusion

In this paper we discuss verification of BOM based composed models at the dynamic semantic level. We propose an extension to the standard BOM, to capture the necessary behavioral details, required to transform it to an executable model such as Colored Petri Nets. We further propose an automatic transformation method to convert E-BOM into our proposed CPN component model, which is useful to represent a model component in executable form (such as CPN) while preserving its structure and behavior. When all components are transformed, we assemble them as a single CPN based composed model using CPN hierarchy tools and analyze it using state space analysis. For the purpose of verification, we propose the modelers to define and verify properties of three types namely: System properties such as deadlock freedom, Goal Reachability and scenario specific (safety or liveness) properties. We also discuss a case study of Field Artillery scenario, and provide its counter example to show how our framework helps to verify a given composition at a dynamic semantic level.

Verification of BOM based composed model facilitates rapid construction and modification of its corresponding federates in HLA based simulations and hence brings forth an improvement in the distributed simulation communities. Colored Petri Nets and its analysis techniques are very useful for accurate and efficient verification as it is one of the competitive formalisms in the specification of the concurrent systems. Their application in the Composability verification proves to be very constructive, especially with a focus on the dynamic semantic composability level. Furthermore, the analysis techniques contributed by the CPN community over a couple of decades provide a significant improvement on efficient and accurate reasoning regarding the model correctness. We are further interested to generalize our approach and specially our CPN component model, to accommodate other component frameworks. We also intend to introduce notions of time in our component model to verify properties that require temporal modalities.

References

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

[2] Ernest H. Page, "Theory and Practice for Simulation Interconnection: Interoperability and Composability in Defense Simulation," in *Handbook of Dynamic System Modeling*.: Chapman & Hall, 2007, ch. 16.

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

[4] Ernest H. Page and Jeffrey M. Opper, "Observations on the complexity of composable simulation," in *Proceedings of the Winter Simulation Conference.*, NJ, 1999, pp. 553–560.

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

[6] Andreas Tolk, "Interoperability and Composability," in *MODELING AND SIMULATION FUNDAMENTALS Theoretical Underpinnings and Practical Domains*.: John Wiley, 2010, ch. 12.

[7] Andreas Tolk, Saikou Y Diallo , and Charles D Turnits, "Applying the Levels of Conceptual Interoperability Model in Support of Integratability, Interoperability, and Composability for System-of-Systems Engineering," *Journal on Systemics, Cybernetics and Informatics*, vol. 5, no. 5, pp. 65-74, 2007.

[8] Brahim Medjahed and Athman Bouguettaya, "A Multilevel Composability Model for Semantic Web Services," *Journal of IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, July 2006.

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

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

[11] Mikel D. Petty, "Verification and Validation," in *Principles of Modeling and Simulation*.: John Wiley & Sons, 2009, ch. 6.

[12] SISO, "Base Object Model (BOM) Template Specification," Simulation Interoperability Standards Organization, Orlando, Florida, SISO-STD-003-2006, 2006.

[13] Paul Gustavson and Tram Chase, "Using XML and BOMS to rapidly compose simulations and simulation environments," in *Winter Simulation Conference*, Washington, DC, 2004.

[14] Paul Gustavson, "Guide for Base Object Model. Use amd Implemenration," Simulation Interoperability Standard Organization (SISO), 2006.

[15] Imran Mahmood, Rassul Ayani, Vladimir Vlassov, and Farshad Moradi, "Behavioral Verification of BOM based composed models," in *22nd European Modeling & Simulation Symposium*, Fes, Morocco, Oct, 2010.

[16] Imran Mahmood, Rassul Ayani, Vladimir Vlassov, and Farshad Moradi, "Fairness Verification of BOM-Based Composed Models Using Petri Nets," in *IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*, Nice, France, June 2011.

[17] Claudia Szabo and Yong Meng Teo, "An Approach for Validation of Semantic Composability in Simulation Models," in *Principles of Advanced and Distributed Simulation, 2009. PADS '09*, New York, 2009.

[18] V S Alagar and K Periyasamy, "Extended Finite State Machine," in *Specification of Software Systems, 2nd edition*.: Springer, 2011, ch. 7.

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

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

[21] CPN Tools. [Online]. http://cpntools.org/

[22] Béatrice Bérard et el, *Systems And Software Verification Model-Checking Techniques and Tools*.: Springer, 2001.