Fairness Verification of BOM-based composed models using 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. However it poses daunting challenges to various issues in research such as *Reusability* and *Composability* in model integration. Various methodological advances in this area have given rise to the development of different component reusability frameworks such as BOM (Base Object Model). However, lack of component matching and support for composability verification and validation makes it difficult to achieve effective and meaningful reuse. For this reason there is a need for adequate methods to verify and validate composability of a BOM based composed model. A verified composed model ensures the satisfaction of desired system properties. Fairness, as defined in section II, is an important system property which ensures that no component in a composition is delayed indefinitely. Fairness in a composed model guarantees the participation of all components in order to achieve the desired objectives.**

In this paper we focus on verification and propose to transform a composed BOM into a Petri Nets model and use different analysis techniques to perform its verification. We propose an algorithm to verify fairness property and provide a case study of a manufacturing system to explain our approach.

Keywords*—*Model Verification;Composability;BOM framework; Petri Nets Analysis; Fairness; Manufacturing System.

I. INTRODUCTION

In the last two decades, the defense industry has invested significant resources in technologies and methods for making independently developed simulations work together [1]. The defense industry gained substantial experiences in interconnecting various simulation systems and the simulation research community has developed some supportive theories under the rubric of simulation composability [1]. The main driving factors behind composability are to enable reuse of existing solutions, cost reductions and cross-domain solutions [2].

The recent discussion on concepts of composability was reignited by Petty's and Weisel's publications on theory of composability [3] according to which, “Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements”. Component-based software engineering has been identified as a key enabler in the construction of composable simulations [4]. At the level of an abstract model, composability is the creation of a complex model from a collection of basic reusable model components [5]. Composability is an effective way to achieve reusability therefore at the model level reuse relies on a composition framework that provides features for both composability and the mapping of composite models into executable form.

Base Object Model (BOM) is a component architecture based on these specifications. It contributes to conceptual modeling by providing the needed formalism and influence the ability to develop and compose model components [6] [7]. BOM is a Simulation Interoperability Standards Organization (SISO) standard. BOM encapsulates information needed to describe a simulation model using XML notation. BOM was introduced as a conceptual modeling framework for HLA (High Level Architecture) which is an IEEE standard 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 is an essential part of BOM that provides means to formalize the component behavior and is our focus in this paper. Without the loss of generality we assume in this paper that a BOM component represents one entity. For details interested readers should refer to [8] [9].

The BOM framework poses an adequate potential for effective model composability and reuse; however it lacks means to express necessary elements of semantic accordance (agreement on the understanding of mutual communication) and behavioral coherency (having an interaction consistent with the common goals) between the composed components, which are essential for reasoning about the validity of the composition [10]. This fact leads us to the investigation of external methods for the matching and verification of BOM composability.

In modeling and simulation, verification is typically defined as the process of determining whether the model has been implemented correctly [11]. Actually, verification is concerned with the accuracy of transforming the model’s requirements into a conceptual model and the conceptual model into an executable model [12]. We focus on the former part and assume that the behavioral correctness is a part of the model’s requirements. In our case the conceptual model is represented by a composed BOM. Our task is to verify that the BOM based composed model satisfies the behavioral requirements such as avoiding deadlock and live-lock or guaranteeing fairness. These requirements are defined in form of system properties and may also include specific reachability properties representing certain desirable or undesirable incidences in the system. All these properties can generally be grouped as *Safety* or *Liveness* requirements. In *Composability Verification* we assess that the model components are correctly assembled such that they satisfy the given requirement specification and their combined behavior is suitable to reach given objectives.

Fairness property as defined in the next section, has a significant place in requirement specifications, and the motivation behind the notion of verifying fairness in a composed model is to disallow infinite executions of some components due to which others are unable to proceed or make progress [13]. It is possible that a deadlock-free composed model makes progress but it cannot guarantee the fulfillment of desired objectives because one of the

components, whose participation is essential, may not get a chance to involve in the interaction. However if a composed model satisfies fairness property, we can guarantee that all components will get a chance to interact and thus their combined behavior influences in a positive way the ability to reach the desired objectives.

In order to achieve suitable composability, the correctness of a composed model is evaluated in various levels of consistencies namely *syntactic*, *semantics* and *pragmatics* [14]. *Syntactic consistency* means that two models can fit together, i.e., the output of one can be read as an input to the other, whereas *Semantic consistency* is concerned with the meaningful interaction of the components. It is further divided into *Static-Semantic* and *Dynamic-Semantic* sublevels. The former involves in having a concise and mutual understanding of the data exchanged by the components participating in the composition, while the latter deals with having a conforming behavior towards their collective objectives [10]. *Pragmatic consistency* on the other hand refers to a context based meaningful interaction of the models [14].

Based on different consistencies involved in the process of model composition and the fact that the BOM framework lacks built-in verification techniques to evaluate them, 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 consistencies [10]. Another approach covers different aspects of validation of semantic consistency in composability of components [15]. A similar work [16] suggests an instrumentation technique to specify verification criteria as a *Model Tester* and check dynamic semantic composability. Using the model tester, various system properties can be modeled as a state-machine component and attached to the composition like a device tester. Another method [17] explores the possibility of using Petri Nets (PN) formalism for the dynamic-semantic composability verification. In this approach a BOM based composed model is transformed into a single PN model and PN Reachability analysis are applied to verify the model behavior [17]. In another similar work presented in [18], an automatic conversion of an architecture created using UML to a Colored Petri Nets based executable model is suggested for determining the potential behavior and performance of the system.

In this paper, we revisit the verification of BOM composability using Petri Nets formalism [17], and utilize certain PN properties analysis methods to verify fairness in the composed model. Here we only consider classical Petri Nets leaving consideration of Timed, Hierarchical and Colored PN extensions to our future work. We present theorems and PN properties required in the fairness verification, and provide a fairness verification algorithm. We also include a case study to explain our approach. The approach to fairness verification presented in this paper can be generalized and applied to verify various other system properties in the given model. It should be noted that the focus of this paper is not to introduce new techniques in Petri Nets; instead we suggest the utilization of existing techniques from Petri Nets community for the purpose of Composability verification. We however propose minor refinements to suit our needs. In our observation, using PN for verification proves to be more fruitful as compared to other similar formalisms, due to the broader range of verification methods and tools and techniques and due to their greater suitability in formal representation of concurrent behaviors.

The rest of the paper is organized as follows: Section II briefly provides basic concepts and definitions of Petri Net formalism that are used in this paper. Section III formulates our Verification approach and presents our algorithm for fairness verification. Section IV provides details on the implementation of our verification process and framework. Section V covers a manufacturing system case study and its counterexample; whereas section VI concludes the paper.

II. PRELIMINARILY CONCEPTS AND NOTATIONS

In this section we briefly discuss basic concepts of Petri Nets theory used in our work. Readers should refer to [18], [19] and [20] for details. We also define and discuss the essentials of fairness property in this section.

A. Petri Nets

Petri Nets (PN) is a graphical and mathematical tool for the formal description of the flow of activities in complex systems. It is particularly suited to represent typical situations like synchronization, sequentiality (producer-consumer problem), concurrency and conflict (mutual exclusion) in a system.

Mathematically a PN is a 5 tuple: $\mathbf{PN = \langle P, T, F, W, M_0 \rangle}$ where:

- P is a finite set of places $P = \{ p_1, p_2, \ldots, p_n \}$
- T is a finite set of transitions $T = \{ t_1, t_2, \ldots, t_n \}$
- F is a set of arcs $F \subseteq (P \times T) \cup (T \times P) \mid P \cap T = \phi$ and $P \cup T \neq \phi$
  (In words: an arc can be connected from place to transition or vice versa but not from place to place or transition to transition).
- W: $F \rightarrow \{1, 2, 3\ldots\}$ is an arc weight function.
- M_0 is the initial marking.

Graphically, a PN is a bipartite graph that has two types of nodes: Circular (or oval) nodes represent places whereas box (or rectangular) nodes represent transitions. These nodes are connected through arcs. A dot in a place represents token. Following are some selected terms and definitions from PN literature that we use in this paper. (It is assumed throughout the paper that a PN has ***m-places*** and ***n-transitions***).

1) Marking

A marking $\mathbf{M: P \rightarrow N}$ is the state of a PN that represents the number of tokens in each place. **M** is a non-negative **m×1** integer-valued vector $\mathbf{M \in N^m}$ where i^{th} component $\mathbf{M}_i$ represents the token load $\mathbf{M(p_i)}$ of the i^{th} place.

2) Arc Weight

Arc weight **w(p,t)** denotes the number of required tokens to be consumed from the input places whereas **w(t,p)** denotes the number of tokens produced to the output place.

3) Enabled Transistion

For a given marking **M** a transition $\mathbf{t \in T}$ is said to be enabled if $\mathbf{\forall p \in {}^\bullet t,\ m(p) \geq w(p,t)}$ i.e, all the input places of transistion **t** have number of tokens greater or equal to the input arc weight.

(Note that $\forall p \in {}^\bullet t$ is the dot notation, which represents all input places that are connected to the transition t. Similarly

$\forall p \in t^{\bullet}$ represents all the output places that are connected to transition t)

4) Firing Count Vector

An **n×1** column vector **X** of nonnegative integers is called firing count vector, where the i^{th} entry of **X** denotes the number of times transistion **t** must be fired to transform $\mathbf{M_0}$ to **M**.

B. Matrix Definitional Form (MDF)

The Matrix Definitional Form (MDF) of a PN consists of the matrices: $\mathbf{A^+}$, $\mathbf{A^-}$, **A**

- $\mathbf{A^+ = [a^+_{ij}]_{nxm}}$ is the *output matrix*, where $\mathbf{a^+_{ij} = w(t_i, p_j)}$; if $\mathbf{p_j \in t_i^{\bullet}}$, and $\mathbf{i \in n}$; $\mathbf{j \in m}$ i.e., if $\mathbf{p_j}$ is connected to the output of $\mathbf{t_i}$ then $\mathbf{a^+_{ij}}$ is equal to the weight of output arc; 0 otherwise.
- $\mathbf{A^- = [a^-_{ij}]_{nxm}}$ is the *input matrix*, where $\mathbf{a^-_{ij} = w(p_j, t_i)}$; if $\mathbf{p_j \in {}^{\bullet}t_i}$, i.e., if $\mathbf{p_j}$ is connected as input to $\mathbf{t_i}$ then $\mathbf{a^-_{ij}}$ is equal to the weight of input arc; 0 otherwise.
- $\mathbf{A = [A^+ - A^-]_{nxm}}$ is the *incidence matrix*. The i^{th} row in **A** denotes the change of the marking as a result of firing transition $\mathbf{t_i}$.

C. State Equation

The incidence matrix **A** is used to compute any reachable marking **M** using the following State Equation:

$$\mathbf{M = M_0 + A.X}$$

Where:

- $\mathbf{M_0}$ is the initial Marking
- **A** is incidence matrix
- **X** is the firing count vector

D. Bounded Petri Nets

Boundedness in Petri Nets is a safety property. A place is bounded with k, if the token count does not exceed k in any marking of a PN. A PN is k-bounded if each place is k-bounded.

E. Invariants

Occurrences of transitions transform the token distribution of a net, but often respect some global properties of markings, regarded as *Linear Invariant Laws*. Invariants are useful for analyzing structural and behavioral properties of Petri Nets. Two most important invariants are the following.

a) P-Invariants

Place Invariants formalize invariant properties regarding places in PN, e.g., if in a set of places the sum of tokens remains unchanged after firing, then this set can define a place invariant. They are useful to evaluate structural properties of PN. P-Invariant of a PN can be formed if there exists an **m×1** vector **Y**, such that for any reachable marking **M**

$$\mathbf{M.Y = M_0.Y \text{ and } A.Y = 0}$$

If there exist a P-Invariant such that $\mathbf{Y(p)>0}$, for all $\mathbf{p \in P}$, then PN is guaranteed to be structurally bounded [20].

b) T-Invariants

Transition Invariants on the other hand formalize properties regarding transition firing sequences applicable to a PN. They are useful to evaluate behavioral properties such as liveliness and fairness. A firing count vector X, is called a T-Invariant if: $\mathbf{A^T.X \geq 0}$

I.e., firing each transition the number of times specified in X, brings the PN back to its initial marking M_0 [20]. There can be multiple T-invariants for a PN, though a minimal T-Invariant is called the *Reproduction vector* of the net [21].

F. Fairness Property

Informally a system is said to be fair if: "*No component of the system that becomes enabled sufficiently often should be delayed indefinitely*" [13]. On the basis of the extent of sufficiency, fairness is generally categorized in the following three types in literature:

a) Unconditional (or Impartial) Fairness

Every component in a system proceeds infinitely often. (Unconditionally)

b) Weak (or Just) fairness

Every component in a system that is enabled continuously from some point onwards eventually proceeds.

c) Strong fairness

Every component in a system that is enabled infinitely often proceeds infinitely often.

Unconditional fairness is also known as non-deterministic choice and is usually present among the components that are independent of each other. A noticeable difference in weak and strong fairness is that weak fairness involves persistent enabling of a component that wants to proceed, whereas strong fairness is not persistently enabled [13].

Some important generalizations of fairness exist in literature:

a) Equi-fairness: To give each component an equal chance to proceed. This type of fairness does not always apply in real world scenarios because of priority policies or some other reasons.

b) Bounded fairness: To give each component an equal number of chances, such that no component proceeds for more then "k-times" without letting the others to take their turn.

In Petri Nets, fairness can be viewed in two perspectives namely: *Transition fairness* and *Marking fairness.* The former corresponds to fairness of choice of transitions, and the latter deals with the fair reachability of states [13]. In this paper we focus on Transition fairness.

III. Verification Approach

In this section, we discuss our approach for the verification of a BOM based composed model that concentrates on the *"Dynamic-Semantic"* consistency and tests that the composition posses correct behavior that satisfies given specifications and objectives. In this paper, we consider "Fairness" as a specification criterion for composability verification. We divide our approach in two main phases: (1) Transformation, (2) PN Analysis, explained below.

A. Transformation

BOM framework uses XML notation for representation. An essential part of BOM is State-machine, which provides means to model the abstract behavior of each component

participating in the composition and is our main concern in the composability verification.

In this phase we transform the state-machines of all members of the composed BOM in to a single PN model. This is an n→1 automatic transformation. In this transformation, we consider each state (of state-machines) as Place in PN and each event (send or receive event) as a Transition in PN such that the sender state **s** and the receiver state **r** (of two different state-machines in BOM) have two places **p** and **q** as input places in PN and connect to a single transition **t** in PN representing the event they exchange. The next states **s´** and r´(after sending or receiving event) have two places **p´** and **q´** in PN as output places such that they have output arcs coming from the transition **t**. We assume that the instances of each component are represented by the tokens (in the initial marking of PN) assigned to the places corresponding to the initial states in the state-machines. The number of tokens (instances) is given as an input parameter.

The transformation process is complete, when all the states and events of every state-machine in BOM are plotted in the PN model such that no element is duplicated, and each place or transition is connected so that there are no broken links. Figure 1 depicts an example of the transformation.

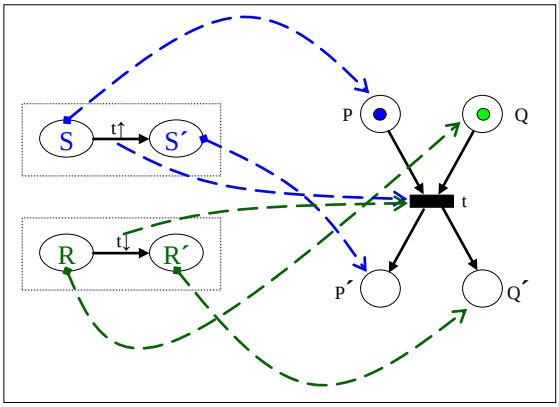

Figure 1. Transformation of BOM state-machines to PN

B. PN Analysis

System properties are explored in this phase depending on the requirements specifications. In Petri Nets, system properties are divided into two groups, namely, structural properties and behavioral properties. Structural properties focus on the structure of the net and are independent of the initial marking; whereas behavioral properties concentrate on the model behavior and depend on the initial marking [18]. Some of the important behavioral properties are *Reachability, Boundedness, Liveness (Deadlock freeness, Livelock freeness)* and *Fairness*. Depending on the nature of the property, various methods have been proposed for verification. These methods include *Algebraic techniques*, *Reachability graph analysis*, *Model Checking* etc. For example, in [17] we have already presented an approach to deadlock detection in a composed model using Reachability graph analysis. In this approach we transformed a composed BOM into PN and explored the Reachability graph for any dead marking. If there exist a dead marking from where no further transition is possible then a deadlock is detected [17].

In this section, we discuss the technique for the verification of fairness property and provide the necessary and sufficient conditions for a PN model to be fair. The evaluation of these conditions in a PN model involves linear algebraic computations; therefore we classify this approach as an *Algebraic technique*. Based on the theorems below, we propose an algorithm for automatic fairness verification.

In Petri Nets, fairness is mainly perceived in terms of occurrences (or proceedings) of transitions. Two transitions $\mathbf{t_1}$ and $\mathbf{t_2}$ are said to be in a fair relation if there exists a positive integer **k** such that for any reachable marking **M** and any firing sequence **σ:**

$$\#(t_1/\sigma) = 0 \Rightarrow \#(t_2/\sigma) \leq k$$

and

$$\#(t_2/\sigma) = 0 \Rightarrow \#(t_1/\sigma) \leq k$$

Where #(t/**σ**) denotes the number of occurrences of transition t in a firing sequence **σ**. In words, neither of the transitions should occur more than a finite number of times (k) without letting the other to do so for at least once. This is known as *bounded fairness* (or *B-Fairness*). If every pair of transition is in a bounded fair relation, then the entire net is said to be fair [21]. For the algebraic verification of fairness in a PN model we rely on the following theorems. Details and proofs of these theorems are discussed in [21].

Theorem I:

Given a PN, if there exists a firing-count vector X, such that: $\mathbf{A^T.X \geq 0}$ and $\mathbf{X \neq 0}$ *then a necessary condition for the PN to be fair is that each entry of* ***X*** *is positive.*

Theorem II:

If a Petri Net ***N*** *is bounded for any initial marking* $\boldsymbol{M_0}$ *then the condition in Theorem I is necessary and sufficient for N to be fair.*

Corollary: If there exist a P-Invariant **Y** *with* $\mathbf{Y(p) > 0}$ *for all* $\mathbf{p \in P}$ *and* $\mathbf{A.Y=0}$*, then the PN is guaranteed to be structurally bounded.*

Theorem III:

A fair Petri Net PN has only one reproduction vector (i.e., a minimal T-Invariant) at the most.

Based on the above definition of fairness and theorems I, II and III, it can be inferred that if there exists a **single** T-Invariant X for a given PN model, i.e., a firing count vector where each entry is non-zero and $A^T.X \geq 0$, and if at least one P-Invariant exists, then we can say that the net is fair.

Based on the above discussion, we propose the fairness verification algorithm presented in Figure 2.

Algorithm 1: Fairness Verification

Input: M_0, A

Result: True/False

```
1 begin
2     ListX_T= Call GetTInvairants()
3     if |ListX_T|=1 and A.X_T≥ 0 and each x in X_T>0 then
4         ListY_P= Call GetPInvairants()
5             if |ListY_P|> 0 then
7                 return true
8             else
9                 return false
10            end if
11    else
12        return false
13    end if
14end
```

Figure 2. Fairness Verification algorithm

In the beginning of the procedure, a sub routine "GetT-Invariants" is called (2) that returns a list of all possible T-invariants of the given PN. If only one T-invariant exists (3) and the multiplication of the T-Invariant with the incidence matrix gives a non-negative result and each entry in the T-invariant $\mathbf{X_T}$ is non-zero then we say that conditions in Theorems I and III are fulfilled. Then we call GetP-Invariant (4). If a non-zero list is returned then it satisfies Theorem II providing that the given PN is structurally bounded thus the given PN is fair. If either or both of tests in (3) and (5) fail, we conclude that the net is not fair.

The methods for calculating P-Invariants (*GetPInvariant*) and T-Invariants (*GetTInvariant*) of a PN model have been extensively studied. The details of these procedures are outside the scope of this paper, however we briefly describe the basic principle to compute the fundamental set of P-invariants and T-Invariants. The principle of finding P-Invariance is presented in Figure 3.

The input of the procedure is the Incidence Matrix **A** and an Identity matrix **B** of size **m×n**. The output is a matrix **B´** whose rows are the fundamental set of P-Invariants. The same procedure is used to find T-invariants but the Incidence Matrix is transposed.

Algorithm 2: P-Invariance (Farkas Method)

Input: A, B /* For T-Invariance A^T*/
Result: B´
1 begin
2 B´ := A|B /* Augmentation of A with n×n identity matrix B */
3 **for** i=1 to n /* n = |T| */
4 **for** each pair of rows b_1, b_2 in B'_{i-1} where $b_1(i)$ and
5 $b_2(i)$ have opposite signs (i.e., annul the i^{th} column)
6 b := $|b_2(i)|.b_1 + |b_1(i)|.b_2$
7 b´ := b/gcd of each element of b
8 augment $B'_{i-1}$with b´
9 **end for**
10 Delete all rows of B'_{i-1}whose i^{th} component is
11 different from 0, the result is B´
12 **end For**
13 Delete the first N columns of B´
14 **return** B´ /*Matrix whose rows contains fundamental P-Invariants*/
15end

Figure 3. P-Invariants calculation algorithm (see [22] and [23] for details) (gcd = Greatest common divisor)

This is the basic principle of calculating Invariance proposed by Julius Farkas in 1902. This method was introduced in the context of PN by J.M. Toudic and later refined by H. Alaiwan and G. Memmi. In this paper we derive the method of Invariance calculations from the much optimized algorithm discussed in [23].

IV. VERIFICATION FRAMEWORK

In this section, we present a composability verification framework as our main contribution. Our proposed framework automates the verification of BOM based composed models and focuses on the dynamic semantic (behavioral) consistency.

Our Java based framework relies on PIPE library [24] as an underlying layer for basic PN operations such as graphical preview and simulation of the PN model (which is not required in the verification process, but it helps to view and study the model). It also utilizes the functions for general matrix operations, constructing A^+, A^- and A matrices, finding P-Invariants and T-invariants etc. We have utilized these functions to implement our suggested verification algorithm. Figure 4 represents our proposed verification framework.

Our framework performs the following steps to verify the BOM composition as marked in Figure 4.

A. Input

A composed BOM XML document is given as an input which is parsed and the components are fetched. Verification requirements (RS) are also provided by the modeler at this step. In this paper, we propose a method for checking only fairness; however similar methods for other properties can be added in the framework as add-on modules in order to provide a wide range of automated tests, allowing the modeler to choose tests according to the verification requirements.

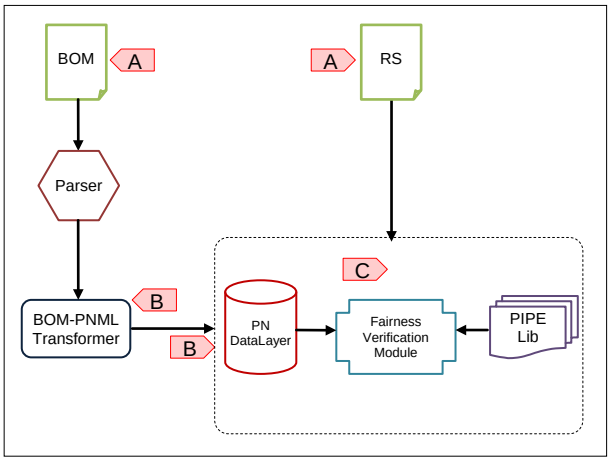

Figure 4. Composability verification framework

B. BOM to PNML Transformation

In this step we transform the composed BOM in to PNML (Petri Net Markup Language) format using our algorithm presented in [17]. PNML is a standard XML-based interchange format for Petri Nets and in our case it is used to represent the transformed composed model in a single place/transition PN model.

In this step, PNML code for Places, Transitions and Arcs is generated. Once the transformation is complete, a consistency check is performed to ensure that there are no broken paths. The output code is written to an XML file representing our composed model in PN format. This file is retaken as input in the framework as PN Data-Layer.

C. Fairness Verification

An automated routine initiates the step mentioned in Algorithm 1 (Figure 2). First, the PN Data layer is accessed and the PN model is populated in a DOM document object making it easy to navigate and access arbitrary data from the PN. Then, A^+, A^- and A matrices are constructed and the initial marking vector M_0 is initialized. Next, we calculate T-invariants of the given PN using functions from the PIPE library. We check that there is only one T-invariant $\mathbf{X_T}$ and all the entries in $\mathbf{X_T}$ are non-zero. Then we check whether the multiplication of A and $\mathbf{X_T}$ returns a null matrix. If all these conditions are fulfilled then we calculate P-invariants. If there exists at least one P-invariant then we are sure that the net is structurally bounded and that the given PN is fair.

V. Case Study

In this section, in order to explain our approach, we discuss a case study of the model of a manufacturing system. This model is composed of three BOM components namely; *Machine-A, Machine-B and a Robot.* Both machines share Robot as a resource that loads raw material on them. Either of the machines can take Robot one at a time. Once the Robot is acquired, it loads raw material on the machine. After the material has been loaded, the machine releases the Robot, proceeds with the production process, and, finally, outputs a finished product. Once the Robot is released, it can be taken by the other machine. Figure 5 shows the components and the workflow of manufacturing system.

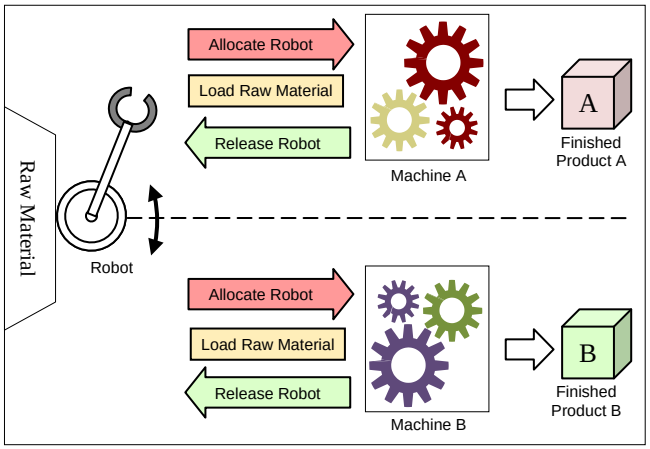

Figure 5. Manufacturing system

All components are built and composed using the BOM framework. We assume for simplicity that the send events are not lost and will eventually be received by the receivers. We also assume that a next send-event is issued only after the previous send-event has been consumed. Figures 6, 7 and 8 depict simplified state-machines of each component involved in the composition. Note that the robot (Figure 8) can send either of the send-events with a random choice.

Given the BOM components of the manufacturing system our objective is to verify that the composed model of the system is fair, i.e., neither of the machines takes the robot more than a certain number of times without letting other to take it at least once.

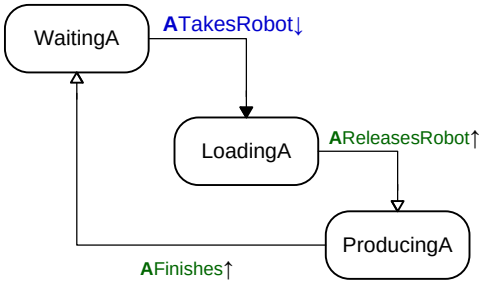

Figure 6. FSM of Machine A

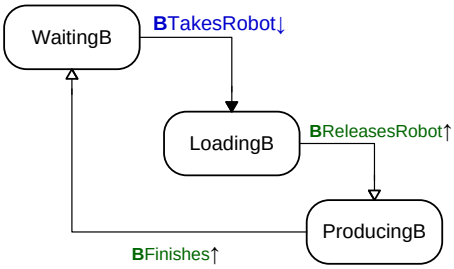

Figure 7. FSM of Machine B

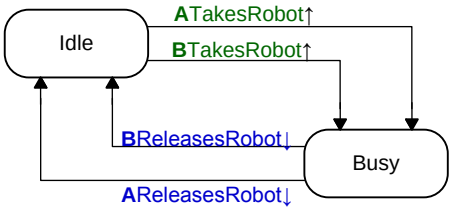

Figure 8. FSM of Robot
[↑=SendEvent] [↓=Receive Event]

The fairness verification is performed as follows.

A. Fairness Verification

In the first step, the BOM xml document is parsed and the state-machine components are fetched. Then, using the BOM-to-PNML transformation algorithm [17], a PNML file is generated. This file is parsed and a DOM object is initialized that acts as a Petri Net Data layer. The required number of tokens is initialized for each place during this transformation step. Figure 9 demonstrates the generated PN model.

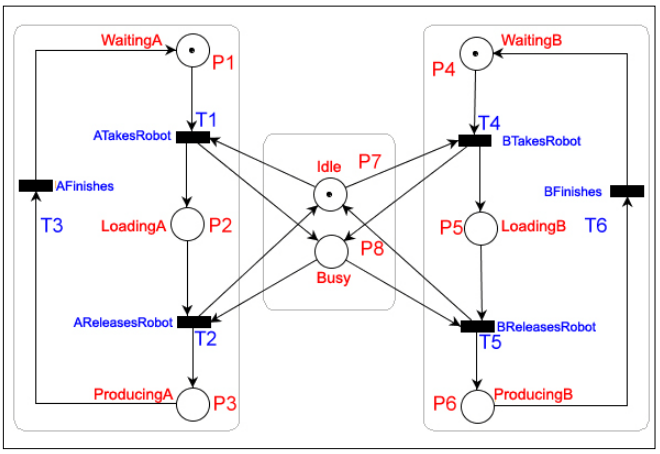

Figure 9. PN Model of the Manufacturing System

In the initialization phase, the initial marking $\mathbf{M_0}$ and the Incidence Matrix **A** are calculated as shown below.

$\mathbf{M_0}$	P1	P2	P3	P4	P5	P6	P7	P8
	1	0	0	1	0	0	1	0

A	P1	P2	P3	P4	P5	P6	P7	P8
T1	-1	1	0	0	0	0	-1	1
T2	0	-1	1	0	0	0	1	-1
T3	1	0	-1	0	0	0	0	0
T4	0	0	0	-1	1	0	-1	1
T5	0	0	0	0	-1	1	1	-1
T6	0	0	0	1	0	-1	0	0

Note that the labels of rows and columns in **A** and elements in $\mathbf{M_0}$ correspond to places and transitions in Figure 9. The matrix **A** is given as input to the Invariant calculation module that detects the following T-Invariants in the PN model of the Manufacturing System:

T1	1
T2	1
T3	1
T4	0
T5	0
T6	0

T1	0
T2	0
T3	0
T4	1
T5	1
T6	1

As there are zero entries in the T-Invariants, so the net is unfair even if there exists any P-Invariant. As the PN is unfair, we cannot guarantee that neither of the machines will over performs by acquiring robot all the time without letting the other to get the robot for at least once. Therefore either of the machines may face a situation in which it is unable to produce enough number of products to meet the required objectives; consequently the composed model may fail to satisfy given specifications.

B. *Counterexample*

In order to understand the fairness verification process, we provide a counterexample. In this example we introduce another component called *Controller* in the composition that can supervise fairness. The job of *Controller* is to enforce fairness in the system. Figure 10 shows the BOM state machine of *Controller*.

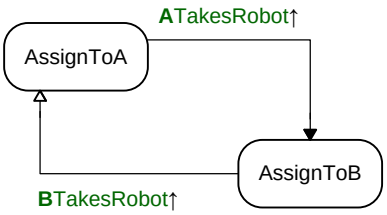

Figure 10. FSM of Controller

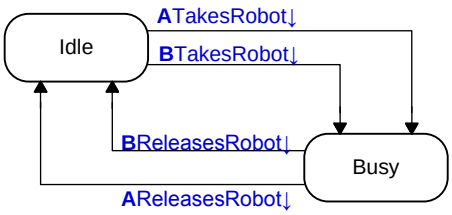

Figure 11. FSM of Modified Robot
Figure 12.

This component regulates the assignment of Robot to machines switching between them. We modify Robot to be a reactive component, which recieves ATakesRobot or BTakesRobot events from the controller, as shown in Figure 11. The PN model of the manufacturing system with added controller, obtained by BOM-to-PNML transformation, is shown in Figure 12.

If we look at the model from the concurrency point of view, we can notice that the *Controller* is acting as a semaphore, which allows both *Machine-A* and *Machine-B* to execute only one at a time.

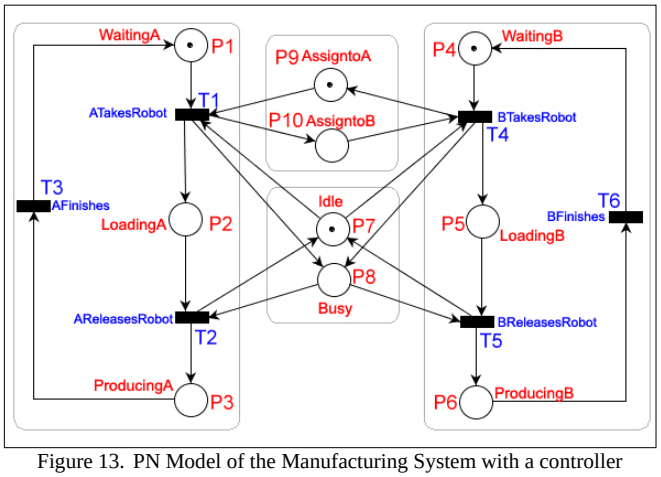

Figure 13. PN Model of the Manufacturing System with a controller

In order to verify fairness property we repeat our verification process for the PN model of counterexample. In the initialization phase, the initial marking $\mathbf{M_0}$ and Incidences Matrix **A** were calculated as follows.

M_0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
	1	0	0	1	0	0	1	0	1	0

A	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
T1	-1	1	0	0	0	0	-1	1	-1	1
T2	0	-1	1	0	0	0	1	-1	0	0
T3	1	0	-1	0	0	0	0	0	0	0
T4	0	0	0	-1	1	0	-1	1	1	-1
T5	0	0	0	0	-1	1	1	-1	0	0
T6	0	0	0	1	0	-1	0	0	0	0

After executing the verification process, we get the following T-Invariant and P-Invariant:

T1	1
T2	1
T3	1
T4	1
T5	1
T6	1

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	1	1	0	0	0	0	0	0	0

Having only one T-Invariant with non-zero entries and having a P-Invariant, satisfies the conditions required for the model to be *1-bounded fair*. Note that the existence of P-Invariant asserts that the net is structurally bounded. Also note that if we assign *k* tokens to the place P9 (Figure12) in the initial marking, the net will become *k-Bounded fair*.

Fairness property becomes significant in the composability verification of a composed model because it ensures the due participation of all components in order to achieve the given objectives. As illustrated by the case study of the manufacturing system, fairness of Robot allocation can ensure that both machines will perform to produce a required number of products. If there is no fairness we cannot guarantee that this objective will be reached.

VI. Summary and Conclusion

In this paper we discuss verification of BOM based composed models. We advocate that transforming a composed BOM into a Petri Nets model and applying existing Petri Nets analysis techniques is a very useful approach for accurate and

efficient verification. We also propose a verification framework that performs the automatic transformation of BOM to Petri nets and based on the requirement specifications executes the verification task. We suggest an approach for verifying fairness property in a PN model and provide an algorithm for it. Subsequently we provide a case study of a manufacturing process to explain and illustrate our approach. This framework can further be extended to accommodate methods to verify other system properties. This approach can be generalized and can be applied for verification of system properties in other component frameworks.

The usage of Petri Nets in the Composability analysis proves to be very useful and promising technique, especially with a focus on the dynamic behavior of the system, as Petri Nets is one of the competitive formalisms in concurrent behavioral representation. Furthermore, the analysis techniques contributed by the Petri Net community over a couple of decades provide a significant improvement on efficient and accurate reasoning regarding the model correctness.

We are further interested to explore methods for the verification of other properties, such as live-lock freeness, starvation freeness, reachability of desirable states. Currently we lack formalism for behavioral verification specification to specify requirements in a formal way. We intend to seek a suitable solution for it and extend our framework in this capacity.

REFERENCES

[1] E. 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.

[2] A. Tolk, "Interoperability and Composability," in Modeling and Simulation Fundamentals Theoretical Underpinnings and Practical Domains.: John Wiley, 2010, ch. 12.

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

[4] R. G. Bartholet, D. C. Brogan, J. . P. F. Reynolds, and J. C. Carnahan, "In Search of the Philosopher's Stone: Simulation Composability Versus Component-Based Software Design," in Simulation Interoperability Workshop, Orlando, 2004.

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

[6] P. Gustavson and T. Chase, "Using XML and BOMS to rapidly compose simulations and simulation environments," in Winter Simulation Conference, Washington, DC, 2004.

[7] P. Gustavson and T. Chase, "Building Composable bridges between the conceptual space and the implementation space," in Proceedings of the winter simulation conference, Washington, DC, USA, 2007.

[8] P. Gustavson, "Guide for Base Object Model. Use and Implemenration," Simulation Interoperability Standard Organization (SISO), 2006.

[9] SISO's Base Object Model (BOM) Specification. http://www.boms.info/standards.htm (Last visited Dec, 2010)

[10] F. Moradi, R. Ayani, S. Mokarizadeh, G. H. A. Shahmirzadi, and G. 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.

[11] O. Balci, "Verification, Validation and Accreditation of simulation models," in Proceedings of the Winter Simulation Conference, Atlanta, GA, 1997.

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

[13] M. Kwiatkowska, "Survey of fairness notions," Information and Software Technology, vol. 31, no. 7, September 1989.

[14] P. Davis, "Composability," in Defense Modeling, Simulation, and Analysis: Meeting the Challenge. Washington, D.C.: The National Academies Press, 2006.

[15] C. Szabo and Y. M. Teo, "An Approach for Validation of Semantic Composability in Simulation Models," in Principles of Advanced and Distributed Simulation, 2009. PADS '09, New York, 2009.

[16] I. Mahmood, R. Ayani, V. Vlassov, and F. Moradi, "Behavioral Verification of BOM based composed models," in 22nd European Modeling & Simulation Symposium, Fes, Morocco, Oct, 2010.

[17] I. Mahmood, R. Ayani, V. Vlassov, and F. Moradi, "Composability Test of BOM based models using Petri Nets," in 22nd IFIP International Conference on Testing Software and Systems, Natal, Brazil, Nov, 2010.

[18] L. W. Wagenhals, S. Haider, and A. H. Levis, "Synthesizing Executable Models of Object Oriented Architectures," in Proceedings of the conference on Application and theory of petri nets: formal methods in software engineering and defence systems, Adelaide, Australia, 2002.

[19] T. Murata, "Petri nets: Properties, analysis and applications 77(4), 541–580 (1989)," in Proceedings of the IEEE, 1989.

[20] M. Silva, Introducing petri nets.: Chapman and Hall, 1993.

[21] H.-C. Yen, "Introduction to Petri Net Theory," in Recent Advances in Formal Languages and Applications.: Springer Berlin, 2006, ch. 25.

[22] T. Murata and Z. Wu, "Fair relation and modified synchronic distances in a petri net," Journal of the Franklin Institute, vol. 320, no. 2, August 1985.

[23] J. Farkas, "Theory of simple inequalities," Journal of Pure and Applied Mathematics , vol. 1902, no. 124, 1902.

[24] M. D'Anna, "Concurrent system analysis using Petri nets: an optimized algorithm for finding net invariants," Computer Communications, vol. 11, no. 4, August 1988.

[25] J. Bloom et al., "Platform independent petri net editor," Imperial College, London, 2007.