2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation

An Approach for Validation of Semantic Composability in Simulation Models

C. Szabo and Y.M. Teo
Department of Computer Science
National University of Singapore
Computing 1, 13 Computing Drive
Singapore 117417
claudias@comp.nus.edu.sg

Abstract

Semantic composability aims to ensure that the composition of simulation components is meaningful in terms of their expressed behavior, and achieves the desired objective of the new composed model. Validation of semantic composability is a non-trivial problem because reused simulation components are heterogeneous in nature and validation must consider various orthogonal aspects including logical, temporal, and formal. In this paper, we propose a layered approach to semantic composability validation with increasing accuracy and complexity. The first layer exploits model checking for logical properties of component coordination including deadlock, safety, and liveness. Next, we address temporal properties by validating composition safety and liveness through simulation time. The third layer provides a formal composition validation guarantee by determining the behavioral equivalence between the composed model and a perfect model. In contrast to state-of-the-art approaches, we propose time-based formalisms to describe simulation components and compare the composition behaviors through time using semantically related composition states.

1 Introduction

Component-based simulation has been of interest in the modeling and simulation research community in recent years [5, 13]. Models developed using reused components are appealing because of shorter development time and their flexibility in meeting diverse user needs [8]. While there are many benefits of component-based modeling, several challenges including semantic composability [5, 8] and semantic composition validation [13] remain.

The validation of composable simulations is a non-trivial problem [8, 13]. Challenges arise from the fact that composition is not a closed operation with respect to validation [1], compositions have emergent properties [4], and reused simulation components were developed in conflicting contexts [13]. Validation must also address *logical* aspects such as deadlock, safety, and liveness, *temporal* aspects such as the behavior of components and compositions over time, and *formal* aspects such as the need to provide a formal measure of the validity of compositions, also called "figure of merit" [5]. In composable simulations, the main validation techniques include formal methods such as the DEVS formalism [15], Petty and Weisel's theory of composability [8], and BOM [6] component abstractions.

We consider a composition to be *valid* and its components to be *semantically composable* if and only if (i) components to be integrated *behave* correctly to form a valid composition both *externally* with respect to their neighbors, and *internally* when safety and liveness properties are preserved over time, and (ii) the resulting composition produces valid output. Constraint validation is the process of verifying the communication of connected components for semantic correctness [12].

This paper presents a new three-layer approach to the validation of simulation compositions, considering logical, temporal, and formal aspects of validity. Each validation layer exploits semantic knowledge of the components to validate different aspects of semantic composition validity. We exemplify our approach using the CoDES (COmposable Discrete-Event scalable Simulation) [12] framework. The contributions of this paper include:

1. An incremental three-layer approach with increasing validation accuracy but with a higher overhead. The first two layers, *Concurrent Process Validation* and *Meta-Simulation Validation*, exploit model checking validation by exposing logical properties including *deadlock*, *safety*, and *liveness* and respectively, tem-

poral properties by validating these logical properties through simulation *time*. *Perfect Model Validation* formalizes the similarity of the composed model with reference to a perfect model. In contrast to current approaches [8], we validate the composition of states over time.

2. An important aspect of validating semantic composability is to establish a formal measure of the composition validity. We introduce a novel semantic metric relation, V_ϵ, for comparing simulation executions of a composed model with a perfect model. Based on well-defined concepts in a component-based ontology, V_ϵ quantifies state similarities and considers only composition states that are semantically related. Through V_ϵ, the formal guarantee to validity has higher credibility compared with current measures [8, 14] because the comparison is done based on both time and semantics, two important considerations in simulation.

This paper is organized as follows. Section 2 presents an overview of the CoDES framework. We describe our three-layered approach in Section 3. In Section 4, we illustrate our approach using a simple example and discuss implementation details. We compare and contrast our approach with current approaches in Section 5. Concluding remarks and future work are given in Section 6.

2 CoDES Framework Overview

CoDES (Composable Discrete-Event scalable Simulation) [12] is a hierarchical component-based modeling and simulation framework. A CoDES component is modeled as a meta-component, an abstraction of the actual component implementation, provided by the component developer when creating the component. The meta-component describes the *attributes* and *behavior* of a component and is used in the framework for model discovery and in the verification and validation of syntactic and semantic composability. The component behavior describes the data that it receives and outputs as a set of states. The transitions between states are defined as a set of triggers expressed in terms of input, time and conditions. More formally, a component C_i is represented by the tuple:

$$C_i = \langle R, A_i, B_i \rangle$$

where R denotes mandatory attributes that are common to all components, A_i denotes component specific attributes, and B_i represents component behavior as a state machine. A component behavior is represented as follows:

$$[I_l]S_p[\Delta t] \xrightarrow{Cond_n} S_t[O_l][A_m]$$

where I_l is the set of input data; S_p is the current state; Δt is the transition duration; $Cond_m$ defines the condition(s) for the state transition; S_t is the next state; O_l is the set of outputs after the state change; A_m is the set of modified attributes after the state change.

Reusable CoDES components are divided into three categories. *Base components* are well-defined atomic entities specific to an application domain. We assume that for each base component type (e.g. *Source* in Queueing Networks Application domain) there exist different base component implementations in the repository (e.g. *SourceOpen* - a *Source* component for open queueing network systems). A developed simulation model is reused as a *standalone simulator* or as *model components* in a larger simulation model. In the adopted component-connector paradigm, components are black-boxes linked by connectors which are responsible for message passing. Composition grammars [10] determine the syntactic composability of simulation components. COSMO, a component oriented ontology, and COML, a markup language that models CoDES components as model meta-components, facilitate component discovery and semantic validation of compositions [12].

3 Proposed Approach

Figure 1 shows our layered approach to semantic validation with increasing accuracy and complexity. In the first layer, the composed model is abstracted as a composition of concurrent processes and desired logical properties are validated by a model checker. In the second layer, a meta-simulation of the composition is executed over time to validate safety and liveness. The third layer, perfect model validation, is a formal but more complex guarantee to validation. The validation in each layer exploits the results or guarantees provided by the previous layers. Major abstraction trade-offs and drawbacks in one layer are addressed in the subsequent layers. All layers assume that the component communication is correct and meaningful [12].

Our proposed approach aims to provide a formal measure of the validity of a simulation, and at the same time considers logical and temporal aspects of the composition. It would be difficult if not impossible to deal with all aspects (formality, logical, temporal) in a single layer. As such, our validation process starts with high level abstractions before adding greater level of detail and realism in the subsequent layers. In the first layer, time is ignored and state machines are compacted as much as possible. The next layer considers time, detailed state machines, and all attribute values. Lastly, the third layer exploits results from the previous layers to provide a more formal, comparable measure of validity.

3.1 Concurrent Process Validation

In *Concurrent Process Validation*, the logical correctness of component coordination is validated. This layer guarantees that *safety* and *liveness* properties hold for any possible

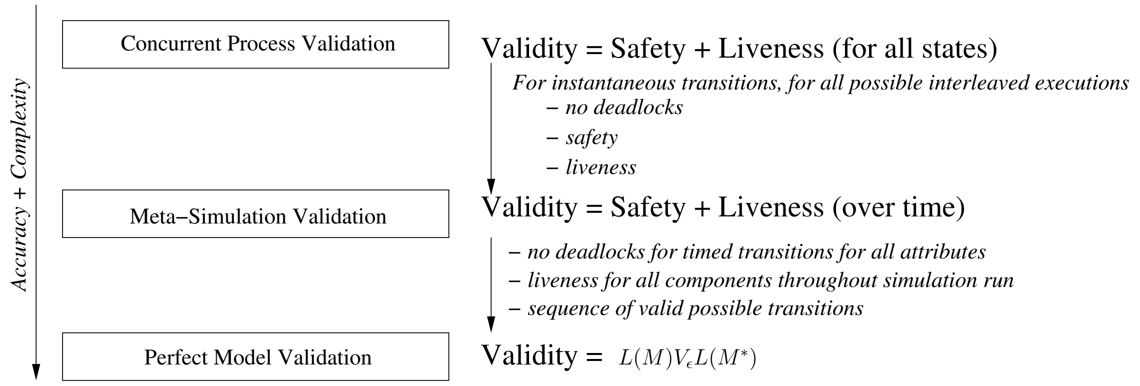
Accuracy + Complexity
Concurrent Process Validation
Validity = Safety + Liveness (for all states)
For instantaneous transitions, for all possible interleaved executions
– no deadlocks
– safety
– liveness
Meta–Simulation Validation
Validity = Safety + Liveness (over time)
– no deadlocks for timed transitions for all attributes
– liveness for all components throughout simulation run
– sequence of valid possible transitions
Perfect Model Validation
Validity = $L(M)V_cL(M^*)$

Figure 1. Layered Approach to Semantic Validation

interleaved execution of the concurrent processes. Furthermore, if deadlock is possible, we check that the composed model is deadlock free in the context of *instantaneous* transitions. Following Lamport's definition, *safety* means that "nothing bad will ever happen", while *liveness* means that means that "something good will eventually happen" [7]. In the context of model checking in our proposed approach, *safety* means that the component does not invalidate some logical properties, and *liveness* guarantees the component reaches a specific pre-defined state in *all* execution traces.

The behavior of each meta-component modeled as a state machine is translated into a logical specification describing the composition together with the desired safety and liveness properties. To prevent state explosion each component state machine is reduced by considering only communication states and attributes that influence state transitions. The actions of non-communicating states are abstracted as a single atomic operation. Similarly, time is not modeled and transitions are considered instantaneous. Furthermore, because the component communication is meaningful and correct, the data sent through the communication channels is abstracted as a single message type. The resulting specification is next verified using a model checker. For example, a Promela Converter program translates the topology into a Promela specification [2], and the Promela specification can be validated using the Spin model checker [2].

3.2 Meta-simulation Validation

The *Meta-Simulation Validation* layer validates the composition run through time. The main aim is to validate that the logical properties demonstrated in the previous layer hold throughout the simulation run. State machines of meta-components together with all time delay mechanisms and other participating attributes are run concurrently in a *meta-simulation* to validate properties such as safety and liveness. This layer guarantees that for all sampled runs and for all meta-components, the safety and liveness properties hold through time.

Safety properties are specified through *validity points* provided by the user. A validity point is a connection point in the topology through which a certain type of data must pass. Safety errors are issued if incompatible data flows through the validity points at any point during the meta-simulation run. Liveness is validated by assigning a *transient* predicate to each component. Such a component specific transient predicate guarantees that if it becomes true during the meta-simulation, then it will become false before a *timeout* elapses. The transient predicate is defined such that a change in its truth value signifies a change in the component state. The transient predicate value is ideally given by the component creator in the meta-component, but it could also be deduced from the state machine.

3.3 Perfect Model Validation

While the previous layers validate important properties such as safety and liveness over time, the *Perfect Model Validation* layer provides a formal measure of composition validity. A new semantic distance metric relation measures the *composition validity* between the composition and a perfect composition. This layer depends on the previous layers for deadlock free, safety, and liveness guarantees as well as for sampled time values. The outcome of this layer is a credible and comparable measure of the validity of the composition, obtained through formal mathematical reasoning.

As shown in the simple example from Figure 2, we divide perfect model validation into five main steps, namely *Formal Component Representation*, *Unfolding and Sampling*, *Composition*, *Simulation*, and *Validation*. For illustration, we apply these steps on a single-server queue example consisting of Queueing Networks base components. A more detailed discussion of this example is presented in Section 4. We formally represent a component as a function of states over time in Figure 2(a). We unfold this function over the simulation time using sampled values for the time attributes in Figure 2(b). The mathematical functional composability of the component functions is validated using our composability definition in Figure 2(c). If the functions are

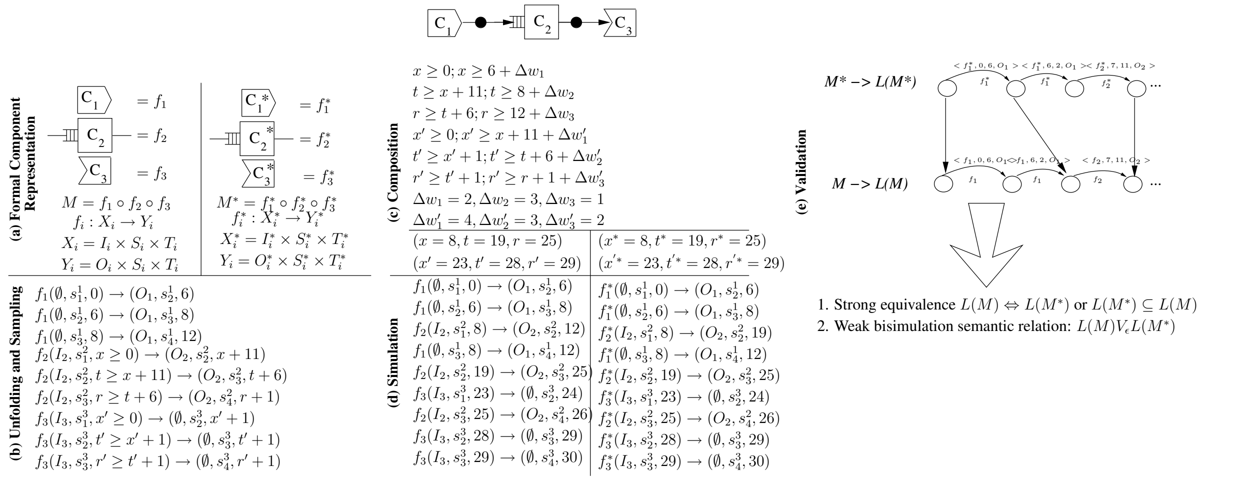
(a) Formal Component Representation
$C_1 = f_1$
$C_2 = f_2$
$C_3 = f_3$
$M = f_1 \circ f_2 \circ f_3$
$f_i : X_i \rightarrow Y_i$
$X_i = I_i \times S_i \times T_i$
$Y_i = O_i \times S_i \times T_i$
$C_1^* = f_1^*$
$C_2^* = f_2^*$
$C_3^* = f_3^*$
$M^* = f_1^* \circ f_2^* \circ f_3^*$
$f_i^* : X_i^* \rightarrow Y_i^*$
$X_i^* = I_i^* \times S_i^* \times T_i^*$
$Y_i = O_i^* \times S_i^* \times T_i^*$
(b) Unfolding and Sampling
$f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$
$f_1(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$
$f_1(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$
$f_2(I_2, s_1^2, x \geq 0) \rightarrow (O_2, s_2^2, x + 11)$
$f_2(I_2, s_2^2, t \geq x + 11) \rightarrow (O_2, s_3^2, t + 6)$
$f_2(I_2, s_3^2, r \geq t + 6) \rightarrow (O_2, s_4^2, r + 1)$
$f_3(I_3, s_1^3, x' \geq 0) \rightarrow (\emptyset, s_2^3, x' + 1)$
$f_3(I_3, s_2^3, t' \geq x' + 1) \rightarrow (\emptyset, s_3^3, t' + 1)$
$f_3(I_3, s_3^3, r' \geq t' + 1) \rightarrow (\emptyset, s_4^3, r' + 1)$
C_1 C_2 C_3
(c) Composition
$x \geq 0; x \geq 6 + \Delta w_1$
$t \geq x + 11; t \geq 8 + \Delta w_2$
$r \geq t + 6; r \geq 12 + \Delta w_3$
$x' \geq 0; x' \geq x + 11 + \Delta w_1'$
$t' \geq x' + 1; t' \geq t + 6 + \Delta w_2'$
$r' \geq t' + 1; r' \geq r + 1 + \Delta w_3'$
$\Delta w_1 = 2, \Delta w_2 = 3, \Delta w_3 = 1$
$\Delta w_1' = 4, \Delta w_2' = 3, \Delta w_3' = 2$
$(x = 8, t = 19, r = 25)$
$(x' = 23, t' = 28, r' = 29)$
$(x^* = 8, t^* = 19, r^* = 25)$
$(x'^* = 23, t'^* = 28, r'^* = 29)$
(d) Simulation
$f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$
$f_1(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$
$f_2(I_2, s_1^2, 8) \rightarrow (O_2, s_2^2, 12)$
$f_1(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$
$f_2(I_2, s_2^2, 19) \rightarrow (O_2, s_3^2, 25)$
$f_3(I_3, s_1^3, 23) \rightarrow (\emptyset, s_2^3, 24)$
$f_2(I_2, s_3^2, 25) \rightarrow (O_2, s_4^2, 26)$
$f_3(I_3, s_2^3, 28) \rightarrow (\emptyset, s_3^3, 29)$
$f_3(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^3, 30)$
$f_1^*(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$
$f_1^*(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$
$f_2^*(I_2, s_1^2, 8) \rightarrow (O_2, s_2^2, 19)$
$f_1^*(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$
$f_2^*(I_2, s_2^2, 19) \rightarrow (O_2, s_3^2, 25)$
$f_3^*(I_3, s_1^3, 23) \rightarrow (\emptyset, s_2^3, 24)$
$f_2^*(I_2, s_3^2, 25) \rightarrow (O_2, s_4^2, 26)$
$f_3^*(I_3, s_2^3, 28) \rightarrow (\emptyset, s_3^3, 29)$
$f_3^*(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^3, 30)$
(e) Validation
M* –> L(M*)
$< f_1^*, 0, 6, O_1 >< f_1^*, 6, 2, O_1 >< f_2^*, 7, 11, O_2 >$
f_1^* f_1^* f_2^* ...
M –> L(M)
$< f_1, 0, 6, O_1 >< f_1, 6, 2, O_1 >$ $< f_2, 7, 11, O_2 >$
f_1 f_1 f_2 ...
1. Strong equivalence $L(M) \Leftrightarrow L(M^*)$ or $L(M^*) \subseteq L(M)$
2. Weak bisimulation semantic relation: $L(M)V_\epsilon L(M^*)$

Figure 2. Perfect Model Validation

composable, then an interleaved execution schedule of all functions is obtained in Figure 2(d). This schedule represents a simulation run of the composition. To validate the composition, we consider that for each type of base component there exists a perfect model in the repository, formally represented as a a perfect function of its states over time. The perfect functions are defined by application domain experts, when the application domain is added to the CoDES framework and are annotated with a star (*) symbol. The perfect functions are composed and a simulation run of the perfect composition is obtained by repeating the above process. We represent the simulation of the composition and the perfect composition using Labeled Transition Systems [9]. In this context, we formally define semantic composability and compare between the two labeled transition systems (Figure 2(e)).

Definition 1 (Formal Component Representation)**.** *The formal representation for a component C_i is a function $f_i : X_i \rightarrow Y_i$, where $X_i = I_i \times S_i \times T_i$, and $Y_i = O_i \times S_i \times T_i$. I_i and O_i are the set of input/output messages, S_i is the set of states and T_i is the set of simulation time intervals at which the component changes state.*

By representing a simulation component as a mathematical function we leverage on Petty and Weisel's formal theory of composability [8]. However, our approach differs by including *time* and *state* as domain coordinates, which allows for a meaningful and detailed definition of a valid model. Based on this, we refine and specialize the formal validation process to a form applicable to environments for component-based simulation development in which time and state are of paramount importance.

The domain of each functional representation is $X_i = I_i \times S_i \times T_i$. I_i represents semantically rich inputs, with correspondents in the COSMO ontology. Next, S_i represents all possible component states, i.e. all values of the component attributes. Lastly, T_i represents the set of simulation time moments at which state transitions occur. The time moment values are the sampled time values that have been used in the Meta-Simulation Validation layer.

The *Unfolding and Sampling* step unfolds the function definition over a period of simulation time using sampled values. For example, the state machine for meta-component C_1 defined as $S_1(\Delta t) \rightarrow O_1 S_1[A_1]$ is unfolded for $\tau = 3$ times with the following sampled values for Δt: $\Delta t = 6, \Delta t = 2, \Delta t = 4$, as the sequence: $f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6), f_1(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8), f_1(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$. We repeat this step for all functions and perfect functions in the composition. For components that require input to proceed, we consider the average time spent by messages in the connectors, depicted in Figure 2 by Δw.

The *Composition* step validates that the functions are mathematically composable.

Definition 2 (Mathematical Composability)**.** *Given f_i and f_j describe adjacent components, with f_j requiring input from f_i and $T_i^{out} = \{t_m^{(i)} | 1 \leq m \leq |O_i|\}$, and $T_j^{in} = \{t_n^{(j)} | 1 \leq n \leq |I_j|\}$, the time values for f_i and f_j when f_i produces output and f_j requires input respectively. Then f_i and f_j are composable iif there exists the bijective binary relation $R = \{(t_n^{(j)}, t_m^{(i)}) \in T_j^{in} \times T_i^{out} | t_n^{(j)} > t_m^{(j)}\}$.*

Informally, all sampled time values for components that require input must be greater than the time moment values for the components that provide them with output. The above definition is the usual mathematical composability definition but only considers time moment values from the three coordinate function domain. The input and output data is already validated in the constraint validation process, and the individual component states are irrelevant at this point in the validation. The time value in-equations can be solved using a constraint solver such as Choco.

In the *Simulation* step, an interleaved simulation run is obtained for model M and for perfect model M^*. The interleaved simulation run orders the function calls based on the time values obtained in the previous step. The simulation runs are represented as Labeled Transition Systems (LTS), $L(M)$ and $L(M^*)$ respectively. Each node represent an annotated composition state given by the tuple $S_{j=1,n*\tau} = [\{state(C_i)_{i=\overline{1,n}}\}, f_{in}, f_{out}]$, where $state(C_i)$ is the state of component C_i, n is the number of components, f_{in} is the function called to enter this node, and f_{out} is the function called for exiting this node. Edges are the function call f_i and f_i^*, and labels are the tuple <*function_name, duration, output*>, where $duration$ represents the function execution time. Our proposed labels consider the $duration$ rather than the $time$ moment when the function begins to execute, because the time moments are already ordered through the directed nature of the LTS.

The *Validation* step is divided into two stages. Firstly, we attempt to prove the equivalence or inclusion between the $L(M)$ and $L(M^*)$ using a *strong* bisimilarity relation [9], in which only the sequence of labels and states is important. This can be done automatically using a bisimulation toolset such as CADP [3]. If we are unable to prove strong equivalence, we relax the constraints in the second stage by defining a semantic metric relation V with parameter ϵ. V_ϵ considers only LTS nodes for which our defined semantic distance is smaller than ϵ. Next, if V_ϵ is a *weak* bisimulation metric relation [9] between $L(M)$ and $L(M^*)$, then $C_1, \dots, C_n$ are semantically composable and $L(M)$ is semantically valid.

Definition 3 (Semantic Parametric Metric Relation)**.** *Let $P \subseteq \{S_1, \dots, S_n\}$, $Q \subseteq \{S_1^*, \dots, S_n^*\}$ a subset of the annotated composition states for $L(M)$ and $L(M^*)$ respectively, with $p \in P$, $q \in Q$, $p = [s(p), f_{in}(p), f_{out}(p)]$, $q = [s^*(q), f_{in}^*(q), f_{out}^*(q)]$, with $s(p) = [state(C_1), \dots, state(C_n)]$ and $s^*(q) = [state(C_1^*), \dots, state(C_n^*)]$ vectors representing component states. We define the semantic relation with parameter ϵ, $V_\epsilon \subseteq P \times Q$, as $V(p,q) = \{(p,q) \in P \times Q \mid \|p-q\|_\sigma \le \epsilon\}$. The semantic vector norm, $\|p-q\|_\sigma$, is defined as*

$$\|p-q\|_\sigma = \frac{DS(s(p), s^*(q)) + \frac{DF(f_{in}, f_{in}^*) + DF(f_{out}, f_{out}^*)}{2}}{2}$$

where $DS(s(p), s^*(q))$ is the semantic distance between composition states, and $DF(f_i, f_j^*)$ is the semantic functional distance between the function names.

The semantic metric relation with parameter ϵ, V_ϵ, determines the semantically related states between the composition and perfect composition LTS. The semantic vector norm has two components, DS and DF. The semantic state distance, DS, measures the semantic differences between component attribute values. The semantic functional distance, DF, determines whether the functions called to enter and exit the LTS nodes are related.

Definition 4 (Semantic State Distance)**.** *Let $s(p) = [state(C_1), \dots, state(C_n)]$, $s^*(q) = [state(C_1^*), \dots, state(C_n^*)]$. The semantic state distance between vectors p and q is defined as*

$$DS(s(p), s^*(q)) = \frac{\sum_{i=1}^{n} |ds(state(C_i), state(C_i^*))|}{n}$$

where

$$ds(state(C_i), state(C_i^*)) = \frac{\sum_{a_i \in A(C_i), a_j^* \in A(C_j^*)} d(a_i, a_j^*)}{m},$$

$A(C_i)$ is the set of attributes for component C_i, $m = |A(C_i)|$ and $d(a_i, a_j^*)$ is defined as

$$d(a_i, a_j^*) = \begin{cases} 0 & \textit{if related}(a_i,\, a_j^*) \textit{ and value}(a_i) = \textit{value}(a_j^*) \\ 0.5 & \textit{if related}(a_i,\, a_j^*) \textit{ and value}(a_i) \neq \textit{value}(a_j^*) \\ 1 & \textit{if } \nexists a_j^* \in A(C_i^*) \textit{ s.t. related}(a_i,\, a_j^*) = \textit{true} \end{cases}$$

where $related(a_i, a_j)$ signifies that a_i and a_j are related in the COSMO ontology.

Definition 5 (Semantic Function Distance)**.** *Let $f_i(p), f_j^*(q)$ the functions called to enter or exit nodes p and q in $L(M)$ and $L(M^*)$ respectively. The semantic state distance DF is defined as*

$$DF(f_i(p), f_j^*(q)) = \begin{cases} 1, & i \neq j \\ 0, & i = j \end{cases}$$

Definition 6 (Semantic Composability)**.** *Let $M = f_1 \circ f_2 \circ \dots \circ f_n$ and $M^* = f_1^* \circ f_2^* \circ \dots \circ f_n^*$ the composition model and perfect model respectively. Then $f_1, f_2 \dots f_n$ are semantically composable iif $f_1 \circ f_2 \circ \dots \circ f_n$ exists (by Definition 2) and V_ϵ (from Definition 3) is a weak bisimulation relation between $L(f_1 \circ f_2 \circ \dots \circ f_n)$ and $L(f_1^* \circ f_2^* \circ \dots \circ f_n^*)$.*

The above definition is similar to that of Petty and Weisel [8]. However, the fundamental difference and our major improvement comes from forcing the weak bisimulation relation to be V_ϵ which we previously defined. V_ϵ is a *semantic* metric relation which considers related composition states according to the COSMO ontology according to a well defined component and attribute hierarchy. By representing components as functions of times and states, $L(M)$ and $L(M^*)$ can be compared based on the timed sequences of component executions. Next, through V_ϵ, the model can be compared with a perfect model based on rigorously defined concepts in an ontology.

4 Example

We demonstrate our approach using a simple single-server queue example presented in Figure 3. Each component has an attached implementation described in the meta-component shown in Table 1. To focus on our approach, we consider only simplified component state machines. More complex examples are discussed in [11].

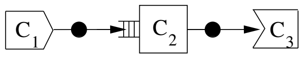

Figure 3. Single-Server Queue Model

4.1 Concurrent Process Validation

Figure 4 shows component state machines translated into a Promela specification. Each state is transformed into a Promela label, which includes input and/or output actions as specified by the meta-component behavior, as well as conditions on attribute values and attribute modifications. Transitions between states are assumed to be instantaneous. Nonetheless, for component C_1 described in process $SOURCE1$ on line 5 we simulate time through the additional process $SourceCounter$ shown on line 9. The role of the $SourceCounter$ process is to modify the inter-arrival time ($interArrivalTime$) until it reaches a predefined value, at which point $SOURCE1$ is activated and produces a message on its out channel. To prevent state explosion, we integrate counter processes only for components that inject jobs into the system. Each connector

```
1   mtype {Job}; chan to1 = [10] of {mtype}; ...
2   hidden byte sourceIAMax = 10; byte sourceIATime; ...
3   proctype CON_ONE_TO_ONE(chan in, out){
4   do :: in ? Job -> out ! Job; od}
5   proctype SOURCE1(int id, noJobsMax; chan out){
6   do :: (sourceIATime == sourceIAMax) -> sourceIATime =0;
7      if :: out ! Job ->
8    progress: printf("[Source] Job sent\n"); fi od }
9   proctype SourceCounter()
10  {do :: (sourceIATime < sourceIAMax) -> sourceIATime++; od}
11  proctype Sink (){...}
12  proctype SERVER3(int id; chan in, out){ S1: {if :: in ?
13  Job -> busy=1;goto S2; fi }
14  S2: {if :: out ! Job -> progress: busy=0; goto S1;}}
15   init {
16  run SourceCounter(); run SOURCE1(1, from1);
17  run SINK1(3, to3 ); run SERVER3(3, to2, from2 );
18  run CON_ONE_TO_ONE(from1, to2); ... }
```

Figure 4. Single-Server Queue Model in Promela

type is defined as a Promela process. For example, process *CON_ONE_TO_ONE* on line 3 describes the one-to-one connector. In the `init` method on line 15, communication channels are assigned to connectors and components based on their connection topology. The Spin model checker validates that there is no deadlock or any unreachable states in the system. To specify liveness, we assign a `progress` label to each component state that produces output.

4.2 Meta-simulation Validation

Our implementation of the meta-simulation validation layer translates the complete state machine of each component into a Java class hierarchy. Attributes and their values provided by the user, state transitions and time are modelled. Next, we construct a meta-simulation of the composed model using the translated classes. During the meta-simulation run, sampling is performed for attributes that require so. This is the case especially for time attributes such as $\Delta interArrivalTime$ in Table 1. The meta-simulation is run for $N = n * noSampling$ times, where n is the total number of components and $noSampling$ is the total number of locations where sampling is done.

From a practical perspective, the simulator developer specifies the *safety* property by describing desired valid output through *validity points* at various connection points in the composition. A validity point contains semantic description of data that must pass through its assigned connection point. For example, the two validity points for the data that passes through the second connector in Figure 3 could be $VP_1 = d_1\{origin = Server, destination = Sink, range = 10; 35, type = double\}$, and $VP_2 = d_2\{origin = Server, destination = Sink, range = 1; 2\}$. A safety error is issued if semantically incompatible data passes through the connection point. *Liveness* is validated by considering a *transient* predicate assigned to each component as shown in Table 1. A component is considered *alive* if its assigned liveness observer has evaluated the transient predicate to *true* and then to *false* in an interval of time smaller than the specified timeout. For example, the transient predicate for component C_2 could be $transient(C_2) = (busy == true)$.

4.3 Perfect Model Validation

In the following we present the detailed validation process only for the selected components C_i represented formally as functions f_i. The same process is repeated for perfect functions f_i^*. For this example, we consider the behavior of the perfect components represented by f_i^* to be the same with respect to input/output transformations to the behavior represented by f_i.

In the *Formal Component Representation* step, the state machine for component C_1 as specified in Table 1 is translated to a formal component representation specified by f_1 which is defined as

$$f_1 : \emptyset \times S_1 \times T_1 \rightarrow \{O_1\} \times S_1 \times T_1,$$
$$f_1(\emptyset, s_i, t) \rightarrow (O_1, s_i', t + \Delta t)$$

where Δt is sampled from a specified distribution and the

	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$
Attribute	$noJobsGenerated = 0$ $timeout = 20$ interArrivalTime: exponential(3) $transient(C_1) : (noJobsGenerated == 1)$	$noJobsServiced = 0$ $timeout = 20$ serviceTime : exponential(6) $busy = false$ $transient(C_2) : (busy == true)$	$noJobsPrinted = 0$ $timeout = 20$ $\Delta printingTime = 1$ $transient(C_3) : (noJobsPrinted == 1)$
Input Constraints	-	$origin = Source\|Server$ $range = 10; 35$ $type = double$	$origin = Server$
Output Constraints	$destination = Server$ $range = 11; 15$ $type = int$	$destination = Server\|Sink$ $range = 10; 20$ $type = double$	-
State Machine	$S_1(\Delta interArrivalTime) \rightarrow S_2$ $S_2 \rightarrow S_1 O_1 [A_1]$	$I_1 S_1 \rightarrow S_2[A_1; A_3]$ $S_2(\Delta serviceTime) \rightarrow S_1 O_1 [A_2]$	$I_1 S_1 \rightarrow S_2$ $S_2(\Delta printingTime) \rightarrow S_1[A_1]$
	$[A_1] = noJobsGenerated + +;$	$[A_1] = (busy = true);$ $[A_2] = (busy = false);$ $[A_3] = noJobsServiced + +;$	$[A_1] = noJobsPrinted + +;$

Table 1. Meta-component Information

function is re-called until $t > T$, where the simulation runs for time $T = 40$ wall clock units.

To obtain specific values for t and Δt, we unfold the function call graph for $\tau = 3$ times and sample the values for Δt, using mean and sampled values from the previous layer, as shown in Table 2.

	Unfold	Δt	**Formula**		
f_1	1	6	$f_1(\emptyset, s_1^1, 0)$	$\rightarrow$	$(O_1, s_2^1, 6)$
	2	2	$f_1(\emptyset, s_2^1, 6)$	$\rightarrow$	$(O_1, s_3^1, 8)$
	3	4	$f_1(\emptyset, s_3^1, 8)$	$\rightarrow$	$(O_1, s_4^1, 12)$
f_2	1	11	$f_2(I_2, s_1^2, x \geq 0)$	$\rightarrow$	$(O_2, s_2^2, x + 11)$
	2	6	$f_2(I_2, s_2^2, t \geq x + 11)$	$\rightarrow$	$(O_2, s_3^2, t + 6)$
	3	1	$f_2(I_2, s_3^2, r \geq t + 6)$	$\rightarrow$	$(O_2, s_4^2, r + 1)$
f_3	1	1	$f_3(I_3, s_1^3, x' \geq 0)$	$\rightarrow$	$(\emptyset, s_2^3, x' + 1)$
	2	1	$f_3(I_3, s_2^3, t' \geq x' + 1)$	$\rightarrow$	$(\emptyset, s_3^3, t' + 1)$
	3	1	$f_3(I_3, s_3^3, r' \geq t' + 1)$	$\rightarrow$	$(\emptyset, s_4^3, r' + 1)$

Table 2. Formal Component Representation

Next, the *Composition* step validates the function composability. Following Definition 2, we obtain constraints for the values of x, t, r and x', t', r' respectively. We consider the time moments at which components produce or require data, as well as the average time spent by messages is the connector queues, $\Delta w_1 = 2, \Delta w_2 = 3, \Delta w_3 = 1$ and $\Delta w_1' = 4, \Delta w_2' = 3, \Delta w_3' = 2$ for f_2 and f_3 respectively:

$$x \geq \Delta w_1, t \geq x{+}11, t \geq 8{+}\Delta w_2, r \geq t{+}6, r \geq 12{+}\Delta w_3 \tag{1}$$

$$x' \geq x{+}11{+}\Delta w_1', t' \geq x'{+}1, t' \geq t{+}6{+}\Delta w_2', r' \geq t'{+}1, \\ r' \geq r + 1 + \Delta w_3' \tag{2}$$

Assume that a solution is:

$$(x = 8, t = 19, r = 25), (x' = 23, t' = 28, r' = 29), \\ (x^* = 8, t^* = 19, r^* = 25), (x^{'*} = 23, t^{'*} = 28, r^{'*} = 29).$$

Interleaved execution schedules are next obtained for both composition and perfect composition, in Figure 5(a)

$f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$	$f_1^*(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$
$f_1(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$	$f_1^*(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$
$f_2(I_2, s_1^2, 8) \rightarrow (O_2, s_2^2, 19)$	$f_2^*(I_2, s_1^2, 8) \rightarrow (O_2, s_2^2, 19)$
$f_1(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$	$f_1^*(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$
$f_2(I_2, s_2^2, 19) \rightarrow (O_2, s_3^2, 25)$	$f_2^*(I_2, s_2^2, 19) \rightarrow (O_2, s_3^2, 25)$
$f_3(I_3, s_1^3, 23) \rightarrow (\emptyset, s_2^3, 24)$	$f_3^*(I_3, s_1^3, 23) \rightarrow (\emptyset, s_2^3, 24)$
$f_2(I_2, s_3^2, 25) \rightarrow (O_2, s_4^2, 26)$	$f_2^*(I_2, s_3^2, 25) \rightarrow (O_2, s_4^2, 26)$
$f_3(I_3, s_2^3, 28) \rightarrow (\emptyset, s_3^3, 29)$	$f_3^*(I_3, s_2^3, 28) \rightarrow (\emptyset, s_3^3, 29)$
$f_3(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^3, 30)$	$f_3^*(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^3, 30)$
(a) Composition	(b) Perfect Composition

Figure 5. Interleaved Execution Schedules

and Figure 5(b). Each interleaved execution is represented as a Labeled Transition System, $L(M)$ and $L(M^*)$ respectively, as shown in Figure 6.

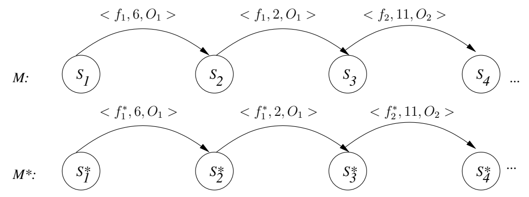

Figure 6. LTS Representation of Model Execution

In the *Validation* step, *strong* equivalence between $L(M)$ and $L(M^*)$ is validated using the BISIMULATOR checker, part of the CADP toolset. For this simple example, the BISIMULATOR returns `true`. As such, there is no need to validate a possible *weak* equivalence by calculating the semantic metric relation V_ϵ. Space constraints prevent us from showing an example of the calculation of V_ϵ.

5 Related Work

Petty and Weisel pioneered a formal theory of composability validation which allows for a composed simulation model to be checked for semantic validity [8]. A composition is modeled as a mathematical functional composition.

The composition simulation is represented as an LTS where nodes are model states, edges are function executions, and labels are model inputs. A composition is valid if and only if its simulation is close by a relation to the simulation of a perfect model. However, time is not modelled and the function representing a component makes an instantaneous transition from input to output. This permits only a *static* representation of the composition. Furthermore, the LTS representation considers the functions strictly in the order they appear in the mathematical composition, which might not be representative of complex compositions. In contrast, we propose a new formal component definition where states change over *time*. Based on this definition, we represent composition simulations as interleaved schedules of component execution, considering the execution duration and output as labels in the simulation LTS. Thus we are able to represent the dynamic change of the entire simulation over time. To provide a more accurate measure of validity, we consider semantically related composition states in the definition of V_ϵ. This would not be possible in the Petty and Weisel approach where a component is abstracted as a one-dimensional integer domain function.

6 Conclusion

We present a three-layer approach for semantic validation of compositions in simulation model integration with increasing accuracy and complexity. The first layer validates the *logical* coordination of composed components by guaranteeing that the composition is free from deadlock and invalid states, and that components are alive. The second layer focuses on composition safety and liveness through time by validating safety properties and component specific transient predicates. Lastly, we propose a formal validation process, by extending current work on formalizing simulation composability. We introduce new formal definitions and propose a novel five-step formal validation procedure. In contrast to current work, a component is represented as a function of its states over *time*, an attribute of paramount importance in simulation. The validation process formally compares the composition execution schedule to that of a perfect composition. This is based on a new semantic metric relation, which considers semantically related composition states. Space constraints prevent us from providing proofs for the time complexity of each layer. However, the validation process is decidable with the first two layers having exponential complexity and the last with polynomial complexity in terms of the number of components. Beside advancing the understanding of semantic composability validation and its complexities, it is also our objective to translate our approach into a practical implementation using existing model checkers and constraint solvers. We have fully automated and integrated the first two layers in the CoDES framework. Base components are translated into the Promela specification and validated using the Spin model checker and perfect model validation is being implemented using the CADP toolset. This paper addresses the semantic validation of simulation model developed using base components. We are extending the validation process to include the more complex reused model components.

References

[1] O. Balci. Verification, Validation and Accreditation of Simulation Models. In *Proceedings of the Winter Simulation Conference*, pages 135–141, Atlanta, USA, 1997.

[2] M. Ben-Ari. *Principles of the Spin model checker*. Springer Verlag, 2008.

[3] H. Garavel. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proceedings of the 19th International Conference on Computer Aided Verification*, pages 158–163, Berlin, Germany, 2007.

[4] R. Gore and P. Reynolds. Applying Causal Inference to Understand Emergent Behavior. In *Proceedings of the Winter Simulation Conference*, pages 712–721, Miami, USA, 2008.

[5] S. Kasputis and H. Ng. Composable Simulations. In *Proceedings of the Winter Simulation Conference*, pages 1577–1584, Orlando, USA, 2000.

[6] 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 *Proceedings of the 11th IEEE Symposium on Distributed Simulation and Real-Time Applications*, pages 145–155, Crete, Greece, 2007.

[7] S. Owicki and L. Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4:455–495, 1982.

[8] M. Petty and E. W. Weisel. A Composability Lexicon. In *Proceedings of the Spring Simulation Interoperability Workshop*, pages 181–187, Orlando, USA, 2003.

[9] J. Srba. On the Power of Labels in Transition Systems. In *Proceedings of the 12th International Conference on Concurrency Theory*, pages 277–291, Aalborg, Denmark, 2001.

[10] C. Szabo and Y. Teo. On Syntactic Composability and Model Reuse. In *Proceedings of the International Conference on Modeling and Simulation*, pages 230–237, Phuket, Thailand, 2007.

[11] C. Szabo and Y. M. Teo. Validation of Semantic Composability in Simulation Models. Technical Report APSTC-TR-2009-01, Department of Computer Science, National University of Singapore, 2009.

[12] Y. Teo and C. Szabo. CODES: An Integrated Approach to Composable Modeling and Simulation. In *Proceedings of the 41st Annual Simulation Symposium*, pages 103–110, Ottawa, Canada, 2008.

[13] A. Tolk. What Comes After the Semantic Web - PADS Implications for the Dynamic Web. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, pages 55–62, Singapore, 2006.

[14] M. K. Traore. Analyzing Static and Temporal Properties of Simulation Models. In *Proceedings of the Winter Simulation Conference*, pages 897–904, Monterey, USA, 2006.

[15] B. Ziegler, H. Prahofer, and T. Kim. *Theory of Modeling and Simulation*. Academic Press, 2000.