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,teoym]@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. Firstly, concurrent process validation exploits model checking for logical properties of component coordination including deadlock, safety, and liveness. Secondly, meta-simulation addresses temporal properties by validating safety and liveness of the composition through simulation time. Thirdly, perfect model validation 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 validation approaches, we propose time-based formalisms to describe simulation components and compare the composition behaviors through time using semantically related composition states. As proof of concept, we discuss examples of queueing networks composition and implementation using existing model checkers and constraint solvers. Lastly, we evaluate the complexity of each layer as a function of the number of components.

1. Introduction

Component-based simulation has been of interest in the modeling and simulation research community in recent years [14, 18, 23]. Simulation composability [17] can be defined as "the capability to select and assemble *simulation components* in various combinations to satisfy *user requirements*". Models developed using reused components are appealing due to their shorter development time and their flexibility in meeting diverse user needs [17]. While the benefits of component-based modeling are appealing, there many

*A version of this paper is published in the Proceedings of the 23^{rd} ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society Press, Lake Placid, NY, USA, pages 3-10, 2009

challenges including syntactic and semantic composability [3, 14, 17], and semantic composition validation [8].

The validation of composable simulations is a non-trivial problem [2, 7, 17, 23]. Challenges arise from the fact that composition is not a closed operation with respect to validation because valid components do not necessary form valid compositions [1]. Next, reused components are developed for different purposes and when composed may result in emergent properties [10]. Similarly, the context in which a reused component was developed and validated might differ from the new context of the composed model [3, 23]. Another challenge arises from the various aspects of component interactions that need to be validated. The validation process must 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" [14]. Because of their orthogonal nature, it is difficult to validate all aspects described above in a single validation process. In composable simulations, the main validation techniques include formal methods such as the DEVS formalism [25], Petty and Weisel's theory of composability [17], and component abstractions such as BOM [15].

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 [22] is the process of verifying the communication of connected components for semantic correctness. It includes validating that messages passed between components are syntactically correct, and semantically meaningful with respect to a communication XML schema and a component-based ontology [22]. Throughout this paper we assume that the communication between components is semantically correct. In the context of this paper, the word "semantic" has two meanings: *ontology related* in which notions are organized in an ontology, and *behavior related* in which the behavior of a component is expressed.

In this paper, we present 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. Our three layer validation process can be employed in a component-based simulation framework that provides a standard component representation capturing component behavior, a component-connector paradigm [21]. We exemplify our approach using the CODES (COmposable Discrete-Event scalable Simulation) [22] 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, the temporal 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 [17], we validate the composition of states over time in both meta-simulation and perfect model validations.

2. An important aspect of validating semantic composability is to establish a formal measure of the validity of the composition. 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 [17, 24] 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 present a theoretical analysis of the time complexity of our approach in Section 5. We compare and contrast our approach with current approaches in Section 6. Concluding remarks and future work are given in Section 7.

2. CODES Framework Overview

CODES (Composable Discrete-Event scalable Simulation) [22] 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. The meta-component describes the *attributes* and *behavior* of a component and is used in the framework for model discovery, and syntactic and semantic validation. Reusable CODES components are divided into three categories in the model repository. *Base components* are well-defined atomic entities specific to an application domain. 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 [21] 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 [22].

The composition of a new simulation model shown in Figure 1 is divided into four stages: *creation* of the conceptual model, *discovery* of reusable components, *selection* of discovered components, and *integration and validation* of the selected components in the new simulation model. A new conceptual simulation model is created using the CODES graphical environment by drag-and-dropping icons representing base and model components on a drawing panel and linking the components using connectors. The *conceptual model* consists of icons representing entities without an attached implementation. Before model discovery, the conceptual model is syntactically verified with respect to its composition grammar. In *model discovery*, a semantic matching index guides the the user to select the best matching component. In *model integration and validation*, the composed model is semantically validated.

A CODES component is modeled as a meta-component, an abstraction of the actual component implementation. The meta-component describes the *attributes* and *behavior* of a component and is used in the framework in 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

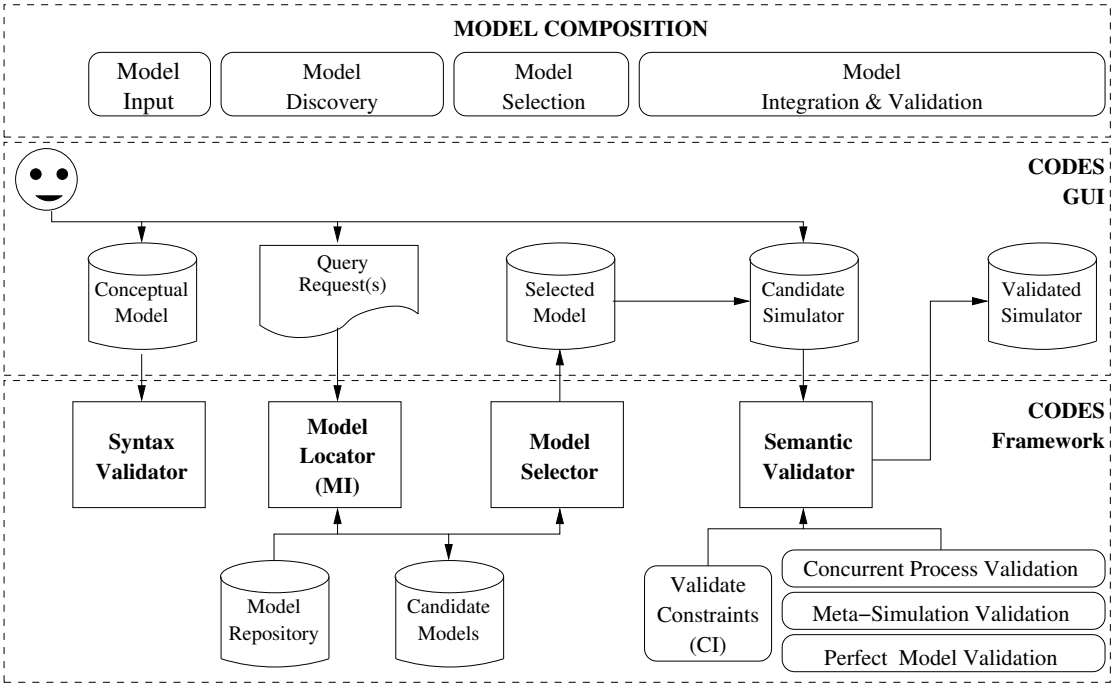

Figure 1. Composable Modeling and Simulation

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.

3. Proposed Approach

Figure 2 presents our layered approach to semantic validation in model integration 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. This is guaranteed by our CODES COML schema and the constraint validation process [22].

Our proposed layered validation 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 these aspects (formality, logical, temporal) in a single layer. As such, our proposed validation process starts from the higher 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

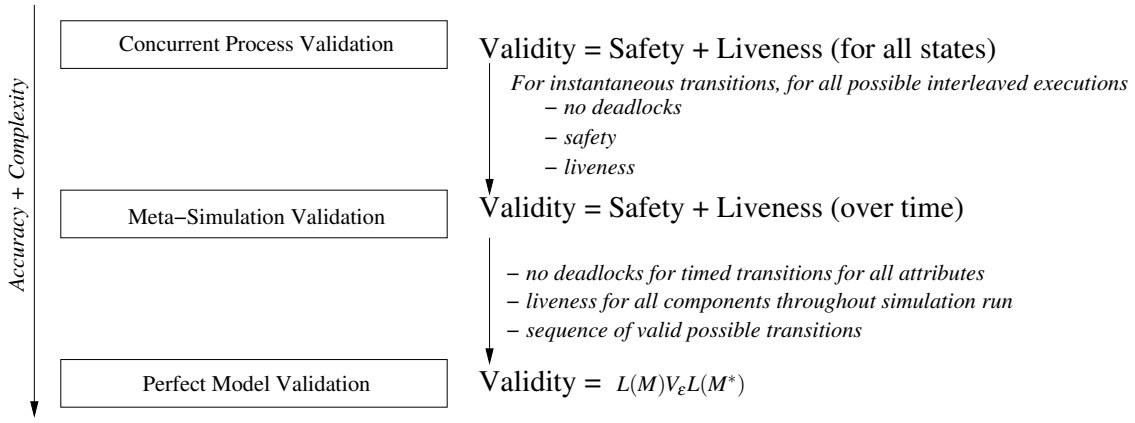

Figure 2. Layered Approach to Semantic Validation

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 interleaved execution of the concurrent processes. Furthermore, we check that the composed model is deadlock free in the context of *instantaneous* transitions. By *safety* we mean that the component does not invalidate some logical properties, and *liveness* guarantees the component reaches a specific pre-defined state in *all* execution traces. However, in application domains such as open queueing networks, deadlock is not inherent and as such need not be validated. We employ model checking techniques to perform our desired validation. A composed model is invalid if it is found to be deadlocked, or if any of the components invalidate their safety or liveness properties. Model checking is employed because it is timeless, automatically performs deadlock verification, and considers all interleaved execution traces.

Figure 3 presents the first layer validation process. The behavior of each meta-component modeled as a state machine is translated into a logical specification using a logic converter module. Different converters can be developed for each application domain and targeting various logical properties. The converter takes as inputs the meta-components and the composition topology. The result is a specification describing the composition together with an expression of the safety and liveness properties. To prevent state explosion each component's 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. This layer assumes that the communication between states is meaningful and correct. Thus the data that is sent through the communication channels is abstracted to a single type of message.

The resulting specification is then verified using a model checker. For example, a Promela Converter can be used to translate the topology into a Promela specification [12], and the Promela specification can be validated using the Spin model checker [4]. In model checking, the entire state space containing all possible interleaved executions of all concurrent processes is analyzed exhaustively for deadlock, safety,

and liveness. For example, in the Promela language we specify various safety properties using simple *assert* statements. Liveness properties (e.g. a component outputs at least one message) are specified using *progress* labels. In the validation process, if the *assert* statement is not verified in any of the states, the model checker issues a safety error. Furthermore, if there exists a cycle that does not visit the *progress* label infinitely often, then a liveness error is issued. An important observation is that discrete time modelling is possible in some specification languages [5]. Unfortunately, these solutions are somewhat obsolete and work only for small systems. A simpler way to model time is by attaching an additional counter process to each main process. However, our results show that the state space explodes even for systems with as less as three components. As such, we integrate counter processes only for components that inject jobs into the system, i.e. for components with only output channels.

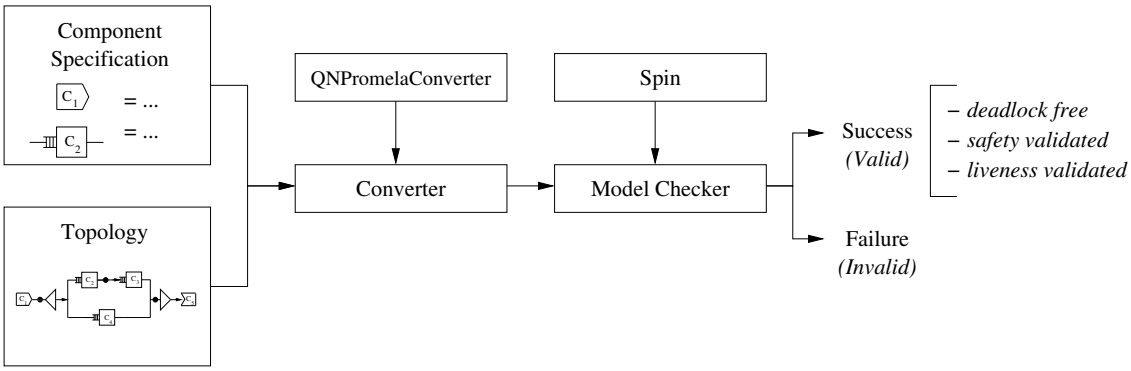

Figure 3. Concurrent Process Validation

The model checker validates that safety, liveness, and deadlock-free properties hold for the abstracted composition. Constraints such as non-floating point types and unbounded queues limit the computational expressivity of the specification but do not hinder the validation of the component coordination. However, state explosion is an important issue. We limit state explosion by considering only important attributes and a limited number of counter processes. Furthermore, artificial termination conditions such as allowing components to process a limited number of messages can be added.

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. This layer assumes that component communication is correct and meaningful and that it is relatively deadlock safe to run the components in parallel.

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 or attributes. The value of the transient predicate is ideally given by the component creator in the meta-component, but it can be deduced from the state machine as well. The timeout interval can be characteristic to the composition or specific to each component.

The state machine specified in the meta-component including attributes and transitions ignored in the previous layer are translated into a class hierarchy and subsequently run. An important class of attributes is time attributes that model time delaying mechanisms (e.g. inter-arrival time, service time). All time attributes are sampled, if necessary, from specific distributions. The distribution type as well as its mean value are also attribute values in the meta-component. Since this method is inherently based on sampling, more than one meta-simulation run is performed with various time attribute values. If any of the meta-simulation returns an error, then the composition is considered invalid.

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. Firstly, when the component functions are unfolded and subsequently composed we assume that deadlock freedom, safety, and liveness are already validated. Secondly, the sampled values of the time attributes are those employed in meta-simulation by the second layer validation process. Furthermore, the interleaved execution schedule is calculated in accordance with the possible correct runs guaranteed by the first two layers. The outcome of the third 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 4, 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 4(a). We unfold this function over the simulation time using sampled values for the time attributes in Figure 4(b). The mathematical functional composability of the component functions is validated using our composability definition in Figure 4(c). If the functions are composable, then an interleaved execution schedule of all functions is obtained in Figure 4(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. 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). The base component implementations may differ from the perfect base component models. Each perfect base component is represented as 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 [20]. In this context, we formally define semantic composability. Based on this definition, we introduce a weak bisimilarity validity relation. The equivalence between the two compositions can be determined using a tool such as CADP [9].

If the two labeled transition systems are equivalent, the composition is considered valid (Figure 4(e)).

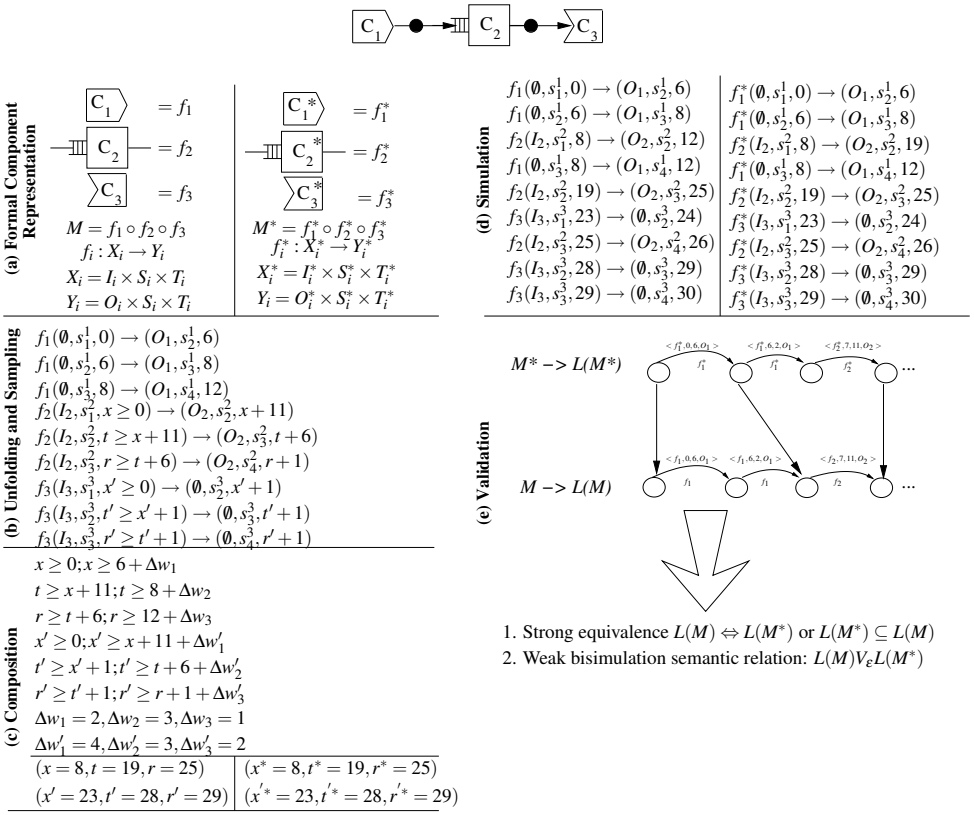

Figure 4. Perfect Model Validation

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 [17]. However, our approach differs by including *time* and *state* as domain coordinates. Although a three coordinate representation adds to the complexity of the validation process, it 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 given by three coordinates, $X_i = I_i \times S_i \times T_i$. The first coordinate, I_i represents semantically rich inputs, with correspondents in the COSMO ontology. Next, S_i represents all possible component states. By component state we mean 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 meta-simulation in the previous 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 is defined as $S_1(\Delta t) \rightarrow O_1 S_1[A_1]$ where Δt is sampled from an exponential distribution with a mean of 4, we have $f_1 : \emptyset \times S_1 \times T_1 \rightarrow \{O_1\} \times$

$S_1 \times T_1, f_1(\emptyset, s_i, t) \rightarrow (O_1, s_i', t + \Delta t)$. For an unfolding coefficient of 3, we obtain the following values for Δt: $\Delta t = 6, \Delta t = 2, \Delta t = 4$, and thus we can write the sequence: $f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6), f_1(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8), f_1(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$, where $f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$ reads as "*f_1 in state s_1^1 with no input at time 0, produces output O_1 at time 6, while changing its state to s_2^1*". 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 4 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 should 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 inequations can be solved using a constraint solver such as Choco [6].

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^* in the simulation run, and labels are the tuple $<$*function_name, duration, output*$>$, where *duration* represents the function execution time. In this form of labelling we consider the *duration* rather than the *time* moment when the function begins to execute, because the time moments at which functions start to execute 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 [16], in which only the sequence of labels and states is important. This can be done automatically using a bisimulation toolset such as CADP [9]. 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 [16] between $L(M)$ and $L(M^*)$, then $C_1, \ldots, C_n$ are semantically composable and $L(M)$ is semantically valid.

Definition 3 (Semantic Parametric Metric Relation)**.** *Let $P \subseteq \{S_1, \ldots, S_n\}$, $Q \subseteq \{S_1^*, \ldots, S_n^*\}$ 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 $s(p) = [state(C_1), \ldots, state(C_n)]$ and $s^*(q) = [state(C_1^*), \ldots, state(C_n^*)]$ vectors representing component states. We define the semantic relation with parameter ε, $V_\varepsilon \subseteq P \times Q$, as $V(p,q) = \{(p,q) \in P \times Q | \|p - q\|_\sigma \leq \varepsilon\}$. The semantic vector norm, $\|p - q\|_\sigma$, is defined as*

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

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 that are called to enter and exit the LTS nodes are related.

Definition 4 (Semantic State Distance)**.** *Let $s(p) = [state(C_1), \ldots, state(C_n)]$, $s^*(q) = [state(C_1^*), \ldots, state(C_n^*)]$. 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

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

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

The above definition is similar to that of Petty and Weisel [17]. 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 in which a well defined component and attribute hierarchy is present. 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. 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 5. Each component has an attached implementation as described in the meta-component [22]. Key meta-component information is shown in Table 1. To focus on our approach, we consider only simplified component state machines.

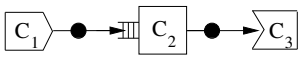

Figure 5. Single-Server Queue Model

	C_1	C_2	C_3
Attribute	$noJobsGenerated = 0$ $timeout = 20$ $time = 200$ $timeScale = 1$ interArrivalTime: exponential(3) $transient(C_1) : (noJobsGenerated == 1)$	$noJobsServiced = 0$ $timeout = 20$ $time = 200$ $timeScale = 1$ serviceTime : exponential(6) $busy = false$ $transient(C_2) : (busy == true)$	$noJobsPrinted = 0$ $timeout = 20$ $time = 200$ $timeScale = 1/5$ $\Delta printingTime = 1$ $transient(C_3) : (noJobsPrinted == 1)$
Input	-	I_1, constraints: $origin = Source\|Server$ $range = 10;35$ $type = double$	I_1, constraints: $origin = Server$
Output	O_1, constraints: $destination = Server$ $range = 11;15$ $type = int$	O_1, constraints: $destination = Server\|Sink$ $range = 10;20$ $type = double$	-
State Machine	$S_1(\Delta interArrivalTime) \rightarrow S_2$ $S_2 \rightarrow S_1O_1[A_1]$	$I_1S_1 \rightarrow S_2[A_1;A_3]$ $S_2(\Delta serviceTime) \rightarrow S_1O_1[A_2]$	$I_1S_1 \rightarrow S_2$ $S_2(\Delta printingTime) \rightarrow S_1[A_1]$
	$[A_1] = noJobsGenerated++;$	$[A_1] = (busy = true);$ $[A_2] = (busy = false);$ $[A_3] = noJobsServiced++;$	$[A_1] = noJobsPrinted++;$

Table 1. Meta-component Information

4.1. Concurrent Process Validation

Figure 6 shows the state machine of every component translated into a Promela specification. Each state is transformed into a Promela label, and the label 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 *SOURCE*1 on line 7 we simulate time through the additional process *SourceCounter* shown on line 11. The role of the *SourceCounter* process is to modify the inter-arrival time (*interArrivalTime*) until it reaches a predefined value. When this happens, process *SOURCE*1 is activated and produces a message on its out channel. Counter processes are introduced for all components that have only output channels.

Each type of connector is defined as a Promela process. For example, process *CON_ONE_TO_ONE* on line 4 describes the one-to-one connector. The fork and join connectors are not part of this composition and as such are omitted. In the `init` method on line 24, communication channels are assigned to the connectors and components according to their connection topology. Similar to the behavior of connectors in the real system, communication in our Promela specification is asynchronous. However, the maximum number of messages in a channel is bounded by a constant value. This is because unbounded queues are not permitted

in the Spin model checker [4], since the focus is on process *coordination* and not computation.

The Promela specification is validated by the Spin model checker. Valid executions can be specified in various ways. By default 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 state in a component that produces output. The Spin model checker validates the system by analyzing all possible process states obtained through the interleaved execution of the active processes. Figure 7 shows the Spin model checker output for the specification presented above. The absence of non-progress cycles is validated. It is important to note the high number of visited states even for this simple example. State space explosion decreases the feasibility of employing this type of validation as a standalone validation process, and thus we include it only as the first layer in our approach.

```
1  mtype {Job}; chan to1 = [10] of {mtype}; chan to2 = [10] of {mtype}; ...
2  hidden byte sourceIAMax = 10; byte sourceIATime; byte noJobsSource = 0;
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 -> progress: printf("[Source] Job sent\n"); fi od }
8  proctype SourceCounter(){do :: (sourceIATime < sourceIAMax) -> sourceIATime++; od}
9  proctype Sink (){...}
10 proctype SERVER3(int id; chan in, out){
11 S1: {if :: in ? Job -> printf("[Server] Job received!\n"); busy=1;goto S2; fi}
12 S2: {if :: out ! Job -> progress: printf("[Server] Job sent! \n"); busy=0; goto S1;}}

14  init {
15 run SourceCounter();
16 run SINK1(3, to3 );
17 run SERVER3(3, to2, from2 );
18 run SOURCE1(1, from1);
19 run CON_ONE_TO_ONE(from1, to2);
20 run CON_ONE_TO_ONE(from2, to3);}
```

Figure 6. Single-Server Queue Model in Promela

```
: ./spin -a promela_spec : gcc -DNP -DMEMLIM=512 -DCOLLAPSE -o pan pan.c
: ./pan -l
(Spin Version 5.1.6--9 May 2008)
Warning: Search not completed
        + Partial Order Reduction
        + Compression
Full statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        non-progress cycles     + (fairness enabled)
        invalid end states      - (disabled by never claim)
State-vector 480 byte, depth reached 979, errors: 0
  9526481 states, stored (1.2111e+07 visited)
```

Figure 7. Spin Model Checker Output

4.2. Meta-simulation Validation

Meta-simulation validation shows that the logical properties demonstrated in the previous layer hold through time. Our implementation 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 inter-arrival time or service time. For example, as shown in Table 1, the inter-arrival time $\Delta interArrivalTime$ for component C_1 is sampled from an exponential distribution with a *mean* of 3. The distribution type and mean values are an example of attribute values provided by the user. Since sampling is performed, 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. If any of the properties does not hold in the meta-simulation runs, the composition is declared invalid.

The most important logical properties that are validated through time are safety and liveness. From a practical perspective, we consider safety to mean that components do not produce invalid output. The simulator developer specifies the desired valid output by providing *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 5 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 anytime during the meta-simulation run semantically incompatible data according to the COSMO ontology passes through the connection point.

Liveness is validated by considering a *transient* predicate assigned to each component. The value of the transient predicate is ideally provided by the component creator in the meta-component as shown in Table 1. Its initial value is `false`. Each component is assigned a *liveness observer* that is notified every time the attributes involved in a transition change values. A component is considered *alive* if its 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. The base components C_i differ from the perfect components C_i^* through additional logging attributes such as $noJobsGenerated$.

In the *Formal Component Representation* step, the state machine for component C_1 as specified in its meta-component is $S_1(\Delta interArrivalTime) \rightarrow S_2, S_2 \rightarrow S_1O_1[A_1]$. This expression 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 function is re-called until $t > T$, where the simu-

lation runs for time $T = 40$ wall clock units.

The above expression is not useful for the *Unfolding and Sampling* step in our approach because during a simulation run t and Δt have specific values. Thus we unfold the function call graph for $\tau = 3$ times and sample the values for Δt, using mean values provided by the user. For component C_1 assume that the inter-arrival time is sampled from an exponential distribution with a $mean = 3$. With sampling and an unfolding grade of $\tau = 3$ we have $\Delta t = 6, \Delta t = 2, \Delta t = 4$. For component C_2 described formally by f_2 assume the service time has an exponential distribution with a mean of $mean = 6$ sampled as $11, 6, 1$. Lastly, we assume component C_3 formalized in f_3 takes 1 unit of time to service each job, so $\Delta t = 1$ for all samples. The values of f_1, f_2, and f_3 are presented in Table 2.

	Unfold	Δt	**Formula**	**Meaning**
f_1	1	6	$f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$	from state s_1^1 with no input at time **0**, to state s_2^1 with output O_1 at time **6**
	2	2	$f_1(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$	from state s_2^1 with no input at time **6**, to state s_3^1 with output O_1 at time **8**
	3	4	$f_1(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$	from state s_1^1 with no input at time **8**, to state s_2^1 with output O_1 at time **12**
f_2	1	11	$f_2(I_2, s_1^2, x \geq 0) \rightarrow (O_2, s_2^2, x+11)$	from state s_1^2 with input I_2 at time **x**, to state s_2^2 with output O_2 at time $\mathbf{x+11}$
	2	6	$f_2(I_2, s_2^2, t \geq x+11) \rightarrow (O_2, s_3^2, t+6)$	from state s_2^2 with input I_2 at time **t**, to state s_3^2 with output O_2 at time $\mathbf{t+6}$
	3	1	$f_2(I_2, s_3^2, r \geq t+6) \rightarrow (O_2, s_4^2, r+1)$	from state s_3^2 with input I_2 at time **r**, to state s_4^2 with output O_2 at time $\mathbf{r+1}$
f_3	1	1	$f_3(I_3, s_1^3, x' \geq 0) \rightarrow (\emptyset, s_2^3, x'+1)$	from state s_1^3 with input I_3 at time $\mathbf{x'}$, to state s_2^3 with no output at time $\mathbf{x'+1}$
	2	1	$f_3(I_3, s_2^3, t' \geq x'+1) \rightarrow (\emptyset, s_3^3, t'+1)$	from state s_2^3 with input I_3 at time $\mathbf{t'}$, to state s_3^3 with no output at time $\mathbf{t'+1}$
	3	1	$f_3(I_3, s_3^3, r' \geq t'+1) \rightarrow (\emptyset, s_4^3, r'+1)$	from state s_3^3 with input I_3 at time $\mathbf{r'}$, to state s_4^3 with no output at time $\mathbf{r'+1}$

Table 2. Formal Component Representation

Next, the function composability is validated in the *Composition* step. Following Definition 2 we obtain constraints for the values of x, t, r and x', t', r' respectively. The constraints on x, t, r derive from the fact that the first call to function f_2 has to take place after at least one call to f_1 has completed and produced output, since f_2 requires output from f_1. Similarly for f_3, the first call has to take place after f_2 has produced *at least one* output. Furthermore, the average time spent by messages in the connector queues is considered. The average time in queue is obtained from the meta-simulation validation layer. Assuming that the average times spent in the connector queues are $\Delta w_1 = 2, \Delta w_2 = 3, \Delta w_3 = 1$ and $\Delta w_1' = 4, \Delta w_2' = 3, \Delta w_3' = 2$ for f_2 and f_3 respectively, the most trivial constraints that can be derived are:

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

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

Next, the constraints are solved by a constraint solver. Assume that a solution is:

$$(x = 8, t = 19, r = 25), (x' = 23, t' = 28, r' = 29).$$

For the perfect functions f_i^*, the constraint solver returns the same solution for (x^*, t^*, r^*) and (x'^*, t'^*, r'^*):

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

The above solutions dictate the interleaved execution schedules of the function calls. Interleaved execution schedules are obtained for both composition and perfect composition. For this simple model in which the component definitions are almost the same with the exception of some attributes, the interleaved schedules presented in Figure 8(a) and Figure 8(b) are the same. Each interleaved execution is represented as a

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

(a) Composition

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

(b) Perfect Composition

Figure 8. Interleaved Execution Schedules

Labeled Transition System, $L(M)$ and $L(M^*)$ respectively, as shown in Figure 9.

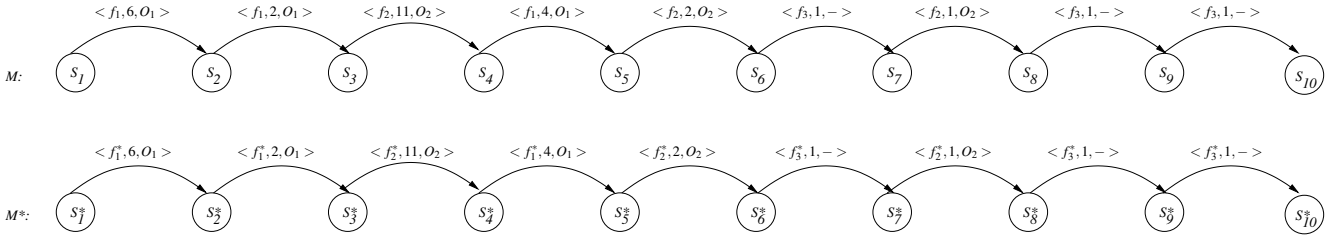

Figure 9. LTS Representation of Model Execution

In the *Validation* step, *strong* equivalence between $L(M)$ and $L(M^*)$ is validated using the BISIMULATOR equivalence 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. For a more complex example, assume that the composition to be validated represents a single-server queue in which the *Source* generates alternatively two classes of jobs that have different service times when serviced by the *Server* component. Space constraints prevent us from showing the calculation of V_ε. However, V_ε cannot be established as a weak bisimulation relation and as such the model is invalid.

The above example raises some interesting issues. Firstly, there is the well known difference between what system experts perceive as *valid* and what can be defined in a computer system as a *valid model* for it to validate independently. In our perfect model validation, a valid model is one that is *close enough* with respect to the states, sequence and duration of component execution, to a perfect model. Yet, what exactly is close enough (i.e. the values of ε), as with all thresholds, remains an open problem. Next is the problem of perfect models and components. While it is acceptable to assume their existence, how they are obtained (for example perfect components could be de-composed from perfect models or they can exist a priori as we suggested in Section 3.3) is still an open question. Lastly, the impact of a different semantic distance *DS* on the weak semantic bisimulation relation remains to be studied.

5. Theoretical Analysis

In this section, we analyze the time complexities of the three validation layers, namely Concurrent Process Validation (O_{CP}), Meta-Simulation Validation (O_{MS}), and Perfect Model Validation (O_{PM}). Space constraints prevent us from presenting the full analysis. Let n be the total number of components and c the

total number of connector branches in the composed model. In Concurrent Process Validation, the entire state space of the composition is exhaustively verified. Thus, the time complexity directly depends on the size of the state space,

$$O_{CP} = O(\mathbb{S})$$

where $\mathbb{S}$ is the size of the state space. This can be divided into two main parts, the components and communication channels that link them. We consider the communication channels separately because they are important entities on which the component synchronization is done. Thus we have

$$\mathbb{S} = \mathbb{S}_{comp} \times \mathbb{S}_{chan} = \prod_{i=1}^{n} T_i \times (\sum_{i=0}^{s} m^i)^q$$

where $\mathbb{S}_{comp}$ is the size of the state space of the components, and $\mathbb{S}_{chan}$ is the size of the state space of the communication channels, each component i has T_i different simulation states, q is the total number of bounded communication channels, allowing a maximum number s of messages, and m is the number of different message types. The component communication is already validated by the Constraint Validation process. Therefore, irrespective of the specification language and model checker used, m can be reduced to a single type of message, i.e. $m = 1$. Considering the worst case where $q = 2c$, and the trivial observation that the number of connectors is a polynomial of grade 1 of the number of components, $c = P^1(n)$, then

$$O_{CP} = O(s^{2P^1(n)} \times \prod_{i=1}^{n} T_i) \tag{3}$$

The right-hand side product is not expanded because its formula depends highly on the type of abstractions employed when the COML component specification is translated into the model checker specification. Nonetheless, the complexity of Concurrent Process Validation is *exponential* in the number of components in the composition.

Intuitively, in the average case the complexity of the Meta-Simulation layer is by design much smaller than O_{CP}. This is because the state space is not exhaustively parsed since the simulation is run for a constrained amount of time and transitions between states are not instantaneous. Furthermore, when considering the liveness of a component, the Meta-Simulation approach considers only two possible truth values of the *transient* predicate. Next, when we validate that components do not stall during their run, the different types of messages that could be in the communication channels are not important. The only significant information about the communication channels is whether they are empty. Thus the complexity of Meta-Simulation Validation becomes:

$$O_{MS} = O(2^{nc}) = O(2^{nP^1(n)}) \tag{4}$$

While O_{MS} is definitely smaller than O_{CP}, the complexity of the Meta-Simulation layer is still *exponential* in the number of components.

The complexity of Perfect Model Validation, O_{PM} is divided into three main parts:

$$O_{PM} = O_{transform} + O_{compose} + O_{bisimulate} \tag{5}$$

where $O_{transform}$ is the time complexity for the formal component representation, unfolding and sampling, and simulation steps (Step 1, 2, and 4); $O_{compose}$ is the time complexity for the Composition step (Step 3)

and $O_{bisimulate}$ is the time complexity for the Validation Step (Step 5). The time complexity for the formal component representation and the unfolding and sampling steps, as well as the simulation step is in the worst case $O(n)$. Thus,

$$O_{transform} = O(n) \tag{6}$$

The time complexity of the Composition step is reduced to the time complexity required by a constraint solver implementation to solve the proposed constraints. The constraint satisfaction problem is NP-Complete. However, the algorithm that solves the particular set of constraints from the Composition step has the time complexity of $O(n)$. This is because we require a single solution and it can be obtained by fixing the values for the time moments for the source components (e.g. x in Equation 1) and propagating the values to the rest of the variables. Therefore,

$$O_{compose} = O(n) \tag{7}$$

For two LTS with N nodes and M transitions, strong and weak bisimilarity between two states can be determined in $O(MN)$ [13]. As such, strong and weak bisimilarity between two LTS can be determined in $O(N^2M)$. For the two LTS that are obtained in the Perfect Model Validation, we have $N = \tau n$ and $M = N - 1 = \tau n - 1$. Thus,

$$O_{bisimulate} = O(\tau^2 \times n^2 \times (\tau n - 1)) = O(n^3) \tag{8}$$

Combining Equations 6, 7, and 8, Equation 5 becomes:

$$O_{PM} = O(n) + O(n) + O(n^3) = O(n^3) \tag{9}$$

Therefore, the complexity of the Perfect Model Validation is *polynomial* in the number of components.

Given n, the number of components in the composition and s, the maximum number of messages allowed in the connectors, we summarize our results in Table 3.

Layer	Complexity
Concurrent Process Validation	$O(s^{2P^1(n)} \times \prod_{i=1}^{n} T_i)$
Meta-Simulation Validation	$O(2^{n \times P^1(n)})$
Perfect Model Validation	$O(n^3)$

Table 3. Theoretical Analysis of the Validation Process

6. 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 [17]. A composition is modeled as a mathematical functional composition. The simulation of a composition 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. In the Petty and Weisel approach, 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 for 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.

DEVS (Discrete Event System Specification) [25] is a formalism derived from general system theory and is designed to describe the structure and behavior of a system. In DEVS, a system is modeled as a blackbox with state, input and output ports. For validation, compositions of DEVS models are represented in the Z specification language [24]. A theorem proving tool based on Z such as Z/EVES [19] is used to verify the model and discover hidden properties. Ambiguities, conflicts and inconsistencies can be discovered in the specification. However, the Z specification language lacks time modeling, a most important attribute in DEVS models. As such, the validation process is incomplete.

A third approach to composition validation [15] uses the Base Object Model (BOM) [11] as a component abstraction. A BOM captures component behavior information including participating entities and their state machines, and information about the possible usage scenarios of the component. This approach assumes that a detailed user specified composition scenario exists to represent a valid composition. The scenario includes the sequence of component execution, as well as events and parameter names for interacting components. Component discovery is done based on the specified scenario. A valid composition of discovered components is one in which the sequence of actions or events is the same as or includes the sequence specified in the scenario. However, the somewhat informal validation process includes the composition and execution of discovered components in *all* possible combinations in order to be compared with the specified scenario. Furthermore, a detailed execution scenario might not be available from the model composer.

7. Conclusion

We present a three-layer approach to the 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. In the third layer, 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. Component functions are unfolded using sampled values. The mathematical composition of the component functions can be validated using existing constraint solvers. Simulations of compositions are then represented as interleaved timed execution schedules. The validation process formally compares the composition execution schedule to that of a perfect composition developed from perfect components in the repository. The comparison determines the equivalence of the schedules based on a new semantic metric relation, which considers semantically related composition states. This is in contrast to Petty and

Weisel's work in which the LTS representation considers function calls in the static order as they appear in the mathematical composition. Our theoretical analysis has shown as expected that the first two layers have exponential complexity. However, we improve the exponential base by employing guarantees provided by prior semantic checks [22]. The exponential base is further reduced by including only relevant properties and by considering only discrete-time increments. The third layer by design has *polynomial* complexity.

In summary, the advantages of our approach are (i) an integrated process that addresses the logical, temporal and formal aspects of validating semantic composability, (ii) validation of state changes over time in the composed model for temporal and formal properties, and (iii) our formal component representation as functions of states over time adds realism to the validation of the composed model against a perfect model, and furthermore, similarity is quantified using our proposed metric called semantic metric relation. 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] J. Banks, J. Carson, B. Nelson, and D. Nicol. *Discrete-Event System Simulation*. Prentice Hall, USA, 2005.

[3] R. Bartholet, D. Brogan, P. Reynolds, and J. Carnahan. In Search of the Philosopher's Stone: Simulation Composability Versus Component-Based Software Design. In *Fall Simulation Interoperability Workshop*, Orlando, USA, 2004.

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

[5] D. Bosnacki and D. Dams. Integrating Real Time into Spin: A Prototype Implementation. In *Proceedings of the International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols*, pages 432–438, Paris, France, 1998.

[6] Choco Constraint Programming System. http://sourceforge.net/projects/choco/, (retrieved Jan. 2009).

[7] P. Davis and R. Anderson. *Improving the Composability of Department of Defense Models and Simulations*. RAND Corporation, 2004.

[8] P. Davis and A. Tolk. Observations on New Developments in Composability and Multi-Resolution Modelling. In *Proceedings of the Winter Simulation Conference*, pages 859–870, Washington, USA, 2007.

[9] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. 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.

[10] 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.

[11] P. Gustavson and L. Root. Object Model Use Cases: A Mechanism for Capturing Requirements and Supporting BOM Reuse. In *Spring Simulation Interoperability Workshop*, Orlando, USA, 1999.

[12] G. Holzmann. *Design And Validation Of Computer Protocols*. Prentice Hall, 1991.

[13] P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Compuation*, 86:43–68, 1990.

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

[15] 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.

[16] D. Park. Concurrency and Automata on Infinite Sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, Karlsruhe, Germany, 1981.

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

[18] M. Pidd. Simulation Software and Model Reuse: A Polemic. In *Proceedings of the Winter Simulation Conference*, pages 772–775, Washinghton, USA, 2004.

[19] M. Saaltink. The Z/EVES System. In *Proceedings of ZUM '97: The Z Formal Specification Notation*, pages 72–85, Reading, UK, 1997.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

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