Towards Semantic Model Composition via Experiments

[Preprint, final version is provided here.]

Danhua Peng
Albert-Einstein-Str. 22
University of Rostock
18059 Rostock, Germany
danhua.peng2@uni-rostock.de

Roland Ewald
Albert-Einstein-Str. 22
University of Rostock
18059 Rostock, Germany
roland.ewald@acm.org

Adelinde M. Uhrmacher
Albert-Einstein-Str. 22
University of Rostock
18059 Rostock, Germany
adelinde.uhrmacher@uni-rostock.de

ABSTRACT

Unambiguous experiment descriptions are increasingly required for model publication, as they contain information important for reproducing simulation results. In the context of model composition, this information can be used to generate experiments for the composed model. If the original experiment descriptions specify which model property they refer to, we can then execute the generated experiments and assess the validity of the composed model by evaluating their results. Thereby, we move the attention to describing properties of a model's behavior and the conditions under which these hold, i.e., its semantics. We illuminate the potential of this concept by considering the composition of Lotka-Volterra models. In a first prototype realized for JAMES II, we use ML-Rules to describe and execute the Lotka-Volterra models and SESSL for specifying the original experiments. Model properties are described in continuous stochastic logic, and we use statistical model checking for their evaluation. Based on this, experiments to check whether these properties hold for the composed model are automatically generated and executed.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis—*Semantic Composability*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Model Composition, Simulation Experiments*

Keywords

Reuse of models; Simulation Experiments; Validation; Semantic Composition

1. INTRODUCTION

Model composition holds the promise of easily assembling complex models from predefined building blocks. Generally, a component is developed as a replaceable part of a system, to be used in unforeseen contexts and for different purposes [39]. Appropriate interface definitions are crucial for this, as they shall provide just the information necessary to (re-)use the component [4]. For many technical areas, libraries of model components have been proven highly effective [8]. Traditionally, model components are viewed as portable building blocks [39] and model composition refers to combining the inputs and outputs of the model components [1]. In non-technical areas like computational biology, however, no libraries of model components could be established yet. Nevertheless, there are public libraries for complete models [18], which are frequently reused as a starting point for model extension or composition [29]. Different approaches towards composing models are distinguished in this context [36]. For example, *Fusion* merges existing models into one model, whereas *Aggregation* creates models by composing models via interfaces.

Unlike model components, which are designed to work in an "unforeseen" context, it is clear that a model designed to answer a specific question of interest — and thus being validated with this question in mind — is not simply reusable in an any context, i.e., cannot simply be composed with another model [3, 25]. Various levels of composability can be distinguished [40], often summarized as syntactic and semantic composability [27]. Semantic composability, which also refers to a model's underlying abstractions and assumptions (this is called conceptual composability in [40]), is particularly challenging to ensure, as these assumptions are rarely stated explicitly and exhaustively. In the following, we try to circumvent this problem by looking at how those assumptions are reflected by simulation experiments and their outcome.

Experiments and their results play a key role in refining and successively enriching models, which is also reflected in workflows that intertwine phases of experimenting and modeling, e.g., [32]. These experiments shall serve as a starting point to tackle the issue of conceptual interoperability and, more generally, the support of modeling by automatic experimentation. Thus, our study does *not* focus on the automatic composition of models and the different manners in which composition can be done, but rather on how to automatically provide important feedback to the user during a modeling process that includes the reuse of different models, i.e., to check whether properties that hold for the reused models also hold for the composed one.

Throughout the paper, we will use Lotka-Volterra models as a running example to illuminate our approach.

This is a pre-print of a SIGSIM-PADS'14 paper published by the ACM. The original version can be found here.
SIGSIM-PADS'14, May 18–21, 2014, Denver, CO, USA.

2. EXAMPLE: COMPOSITION OF LOTKA-VOLTERRA MODELS

The Lotka-Volterra model, one of the classic predator-prey models, is a well-known example of a mathematical model of population biology (as described in [30], it was first presented in [19, 41]). It describes the interaction between populations, i.e., predators and prey, in a rather abstract manner. Here, we focus on a specific variant of the Lotka-Volterra model, where one predator species hunts one prey species and the prey depends on an unlimited supply of food. To avoid an exponential growth of the prey population in the absence of predators, competition among prey is also considered. Let N_1 denote the prey population size and N_2 denote the predator population size, so that the deterministic equations are (cf. [30, p. 176]):

$$\begin{aligned} \mathrm{d}N_1/\mathrm{d}t &= N_1 * (b - k * N_1 - a * N_2) && (prey) \\ \mathrm{d}N_2/\mathrm{d}t &= N_2 * (-d + c * N_1) && (predator) \end{aligned}$$

where b is the birth rate of the prey, d is the mortality rate of the predator, a and c are the interaction coefficients, and k is the competition coefficient.

Different questions can drive the modeling and simulation of prey and predator systems. For example, one may be interested in the stationary states of the system, e.g., the "coexisting state", where both prey and predator populations exist, the "prey state", where the predators die out and only prey survive, or the "empty state", where predators first extinguish the prey population and afterward die out [6]. Similarly, it may also be interesting to compare the size of the predator and prey populations. These questions are reflected in different *hypotheses*, i.e., under which circumstance, e.g., parameter values and initial state, does a certain *statement* or *property* hold.

Assuming we develop a set of Lotka-Volterra models, each characterized by some hypotheses and experiments: what properties of the composed model can be deduced? For example, if we have two Lotka-Volterra models and for each of those the hypothesis "coexisting state" has been shown to hold, can we assume that the hypothesis still holds in a composed model where two predator species hunt the same prey, or two prey species are hunted by the same predator? What does a violation, i.e., a falsified hypothesis, tell us about the conceptual validity of our composition?

In the following, we approach these questions based on concrete simulation studies that focus on three properties. Two properties, "coexisting state" and "empty state", are commonly checked in Lotka-Volterra models [6]. The third, "recovery comparison", states that the prey population will recover faster than the predator population if both populations has been disturbed significantly, i.e., both populations are quite small.

2.1 Ingredients

To test our approach, we need a modeling formalism, an execution algorithm, a way to describe simulation experiments, and a way to describe the model properties that shall be satisfied.

2.1.1 Model Description and Execution

We use ML-Rules [21] to describe the Lotka-Volterra models, which is a rule-based, multi-level modeling language for (cell) biological systems that has been realized for JAMES II [16]. ML-Rules can describe a dynamic hierarchy of nested species, and also supports downward and upward causation across different levels of the hierarchy. For each rule, arbitrary reaction rate kinetics using any kind of mathematical expression and constraints are allowed to specify state transitions in a flexible manner. Several other modeling formalisms supported by JAMES II would also allow to define these comparatively simple models. However, as we aim at applying our technique to concrete cell biological modeling and simulation studies, e.g., to extend the Wnt-pathway model from [22] to include membrane dynamics, we use ML-Rules here as well. We use the reference stochastic simulation algorithm from [21], which is based on Gillespie's approach [11], to simulate the models (instead of the faster but approximative tau-leaping variant [15]).

2.1.2 Experiment Description

SESSL (*Simulation Experiment Specification via a Scala Layer*) is an embedded domain-specific language for simulation experiments [9]. It serves as an additional software layer between users and simulation systems, facilitates the reuse and execution of simulation experiments, and offers various features, e.g., for experiment design and simulation-based optimization. As already mentioned, a variety of experiments are executed during the development of a simulation model, e.g., sensitivity analysis or parameter estimation [32]. The user might wish to store experiments and results together with the model. Models can be annotated with SESSL specifications, which can also be generated. So far, SESSL was focused on the execution of experiments, similar to other approaches for simulation experimentation (e.g., NEDL [14]). It did not allow to specify which model properties shall hold, i.e., what the experiment results should look like. However, an explicit and formal description of the expected outcome is required for our approach.

In general, the motivation of simulation experiments is rarely stated in a formal manner. This is in contrast to verification and model checking approaches, where the property of the system to be checked needs to be formally defined [24]. Consequently, experiments that combine techniques from simulation and verification, e.g., in statistical model checking, include explicit formal statements about the properties of the trajectories that are checked. Nevertheless, SESSL is easy to extend, so we choose to use it for experiments description.

2.1.3 Property Description

Linear Temporal Logic (LTL) is widely used to check the properties of individual trajectories [10, 47], and thus allows to express a broad range of dynamic model properties. To check an output trajectory π, we rely on a JAMES II-based reimplementation of the model-checking algorithm introduced by Fages et al. [10]. The original algorithm captures the following operators of LTL: X (next), G (global), F (finally), and U (until). We extended the algorithm to include the R operator (release) as well (see [2]). However, LTL might not suffice to describe all properties of interest. Thus, we allow to express custom properties via predefined predicates, which must also provide the corresponding algorithms to check them on a trajectory.

ML-Rules is based on Continuous-Time Markov Chain (CTMC) semantics, so that the results are stochastic and multiple replications are required for analysis. For some replications a property may hold, for others it may not hold. Thus, we need to express our expectation regarding replications with probabilities. In analogy to Continuous Stochastic Logic (CSL) [46, 35], which has been proposed as a formalism for expressing properties of CTMC, we define $Pr_{\bowtie p}(\phi)$ with $\bowtie \in \{<, \leq, >, \geq\}$ and an initial state s so that

$$s \vDash Pr_{\bowtie p}(\phi) \iff Prob(\pi \in Path(s) | \pi \vDash \phi) \bowtie p \quad (1)$$

We extended SESSL to support the definition of such probabilistic statements, where ϕ could be an LTL formula or a predefined predicate. In contrast to [46, 35], however, we currently do not support nested probabilities.

Statistical model checking relies on executing stochastic models, and on hypotheses testing. Consider the null hypothesis H_0 that some property does *not* hold in s, and the alternative hypothesis H_1 that the property does hold in s: the probability to accept H_1 although H_0 is true (false positive) should be at most $\alpha \in (0, 1]$, and the probability to accept H_0 although H_1 is true (false negative) should be at most $\beta \in (0, 1]$. α and β are error bounds for statistical model checking, and as it is quite difficult to ensure a low probability for both types of errors, typically an indifference region of size 2δ is defined. Different approaches exist to minimize the amount of needed replications (e.g., see [45]). For our proof of concept, we follow the approach outlined in [35, p. 5–6] and find the smallest number of replications, n, so that the probability of false positives (false negatives) is smaller than α (β) when assuming a binomially distributed number of successful checks for ϕ, with parameters n and $p - \delta$ (n and $p + \delta$). Again, p denotes the assumed probability (see Equation 1).

2.2 The basic Lotka-Volterra Model

Figure 1 shows the parameters, the species, the initial state, and the reaction rules of a Lotka-Volterra model defined in ML-Rules. Please note that predators and prey are modeled as populations whose individuals encounter the events of death or reproduction stochastically. In the original Lotka-Volterra model equations, five parameters (i.e., a, b, c, d and k, see Section 2) as well as the initial predator and prey population sizes can be set by the modeler. To simplify the example, we reduced the parameter space and thus our ML-Rules models do not distinguish between the interaction coefficients, so both have the same value ($a = c$), and we keep the competition coefficient $k = 0.002$ constant.

A model with the fox being the predator and the rabbit being the prey shall illuminate the experiments. As a first property, we tested the "coexisting state", i.e.,

$$G(\#Rabbit > 0 \wedge \#Fox > 0).$$

We assume that both predators and prey will survive, with a probability of 0.8, i.e.,

$$s \vDash Pr_{\geq 0.8}(G(\#Rabbit > 0 \wedge \#Fox > 0)).$$

The experiment is specified in SESSL (see Figure 2) and executed with JAMES II. The parameter value ranges for which the simulation output fulfills this statement are:

```
a: 0.014;
b: 0.6;
d: 0.7;
k: 0.002;
nFood:100;
nPredator:10;
nPrey:100;
Food();
Predator();
Prey();

>>INIT[(nFood) Food + (nPredator) Predator + (nPrey) Prey];

// Prey reproduces
Food:f + Prey:x -> Food + 2 Prey @b*#x;
// Prey dies by competition
Prey:x + Prey:z -> Prey @k*#z*#x;
// Predator reproduces based on successful hunting
Predator:y + Prey:x -> 2 Predator @a*#y*#x;
// Predator dies
Predator:y -> @d*#y;
```

Figure 1: A Lotka-Volterra model described in ML-Rules

```
val exp = new Experiment with Observation with Hypotheses {
  model = "file-mlrj:/./LotkaVolteraFoxRabbit.mlrj"
  scan(
    "a" <~ range(0.010, 0.001, 0.015),
    "b" <~ range(0.6, 0.1, 1.0),
    "d" <~ range(0.1, 0.1, 1.0),
    "nPrey" <~ range(100,100),
    "nPredator" <~ range(10,10))
  stopCondition = AfterWallClockTime(minutes=2) or
       AfterSimTime(10)
  observe("Rabbit","Fox")
  observeAt(range(0.0, 0.1, 10))
  assume(
   Pr(G(variable("Rabbit") > 0 and variable("Fox") > 0)) >= 0.8)
}
```

Figure 2: SESSL experiment specification to test the "coexisting state" hypothesis in a basic Lotka-Volterra model.

$$a : 0.010 - 0.015, b : 0.6 - 1.0, d : 0.5 - 1.0$$
$$nPrey = 100, nPredator = 10$$

These parameter ranges define the initial states s (see Equation 1) for which the property has been shown to hold with the given probability, by statistical model checking. Please note the parameters for the initial values of prey ($nPrey$) and predators ($nPredator$) have been fixed. For simplicity, we set the probability p in all our experiments to 0.8.

Similarly, for the properties "empty state" and "recovery comparison", experiments using the same model are executed. For the "empty state" (i.e., the prey dies out first, then the predators die out), the property ϕ can be expressed in LTL as

$$((\#Prey = 0)\ R\ (\#Predator > 0)) \wedge F\ (\#Predator = 0).$$

Taking the probability (and multiple replications) into account, the property which needs to be tested is denoted as $Pr_{\geq 0.8}(\phi)$. In SESSL, this can be expressed as

```
assume(
 Pr(((Negation(variable("Rabbit") > 0)) R (variable("Fox") > 0)
      ) and (Negation(G(variable("Fox") > 0)))) >= 0.8)
```

From the experiments, we learn that this property holds for the initial states s where

$$a : 0.050 - 0.070, b : 0.1 - 0.4, d : 0.7 - 1.0$$
$$nPrey = 100, nPredator = 30$$

Both, property and parameter assignments, together form our hypothesis regarding the model behavior.

The "recovery comparison" property — if both populations are disturbed at the beginning to have the same small size, the prey population will recover faster — is difficult to be expressed in LTL. Therefore, this property is currently implemented as a predefined predicate (see Section 2.1.3). The initial sizes of both predator and prey population are set to a common low value. In principle, the predicate is evaluated by comparing the time points where the first peak occurs in each population. The earlier the first peak occurs, the faster the population has recovered. However, the population trajectories contain random fluctuations, and many sophisticated methods have been developed to find optima in time series with noise (e.g., [34, 7]). We use a more simple approach, which may not be as rigorous but appears to be sufficient for our purpose. In our method, a certain time point is selected and only the time points between the initial and the selected time point are considered. From those, the time point with the maximum value is identified, and the recovery rate is calculated as the slope between the initial value and this maximum. If the maximum equals the initial value, then the recovery rate is calculated as the slope between initial and selected time point. Although such a manual implementation of predicates is not too difficult in JAMES II, it is not acceptable for all users. Therefore, future work will be aimed at extending the language constructs currently used for describing properties of trajectories.

Again, experiments are executed to check the above property. The hypothesis that the probability of The property "recovery comparison" is satisfied with a probability larger than 0.8 for the following initial states:

$$a : 0.010 - 0.028, b : 0.6 - 1.0, d : 0.5 - 1.0$$
$$nPrey : 10, nPredator : 10$$

2.3 Composing Lotka-Volterra Models

Given two Lotka-Volterra models as shown in Figure 1, i.e., each having two species, these can be composed in three ways: the *same prey* composition, the *same predator* composition and the *food chain* composition.

In the *same predator* composition, one predator species and two prey species constitute the composed model. The predator hunts on both prey species. This type of composition requires that the predators in the two model components are the same. Similarly, in the *same prey* composition, there are two predator species and one prey species in the composed model, and both predator species hunt on the same prey. In the *food chain* composition, the prey in one reused model has the role of the predator in the other model component. Thus, in the composed model, it is a food chain where one species functions as both, prey and predator.

```
1 Food:f + Rabbit:x -> Food + 2 Rabbit @b*#x;
2 Rabbit:x + Rabbit:z -> Rabbit @k*#z*#x;
3 Wolf:y + Rabbit:x -> 2 Wolf @a*#y*#x;
4 Wolf:y -> @d*#y;
5
6 Fox:y + Rabbit:x -> 2 Fox @a*#y*#x;
7 Fox:y -> @d*#y;
```

Figure 3: An example of the same prey composition.

```
1 Food:f + Rabbit:x -> Food + 2 Rabbit @b*#x;
2 Rabbit:x + Rabbit:z -> Rabbit @k*#z*#x;
3 Fox:y + Rabbit:x -> 2 Fox @a*#y*#x;
4 Fox:f -> @d*#f;
5
6 Wolf:w + Fox:y -> 2 Wolf @a*#y*#w;
7 Wolf:w -> @d*#w;
```

Figure 4: An example of the food chain composition.

Similar to the model in Figure 1, we developed and experimented with Lotka-Volterra models for foxes and rabbits, wolves and rabbits, wolves and foxes, and wolves and sheep. Now we wish to compose those. As outlined in [26], the modeler needs to decide during the composition whether a species with the same name refers to the same species in the composed model. If not, renaming is required. For example, if the wolves were hunting another kind of rabbit, its name needs to be changed. Besides renaming species, the modeler could also change the configuration of parameters during composition. In the food chain composition, for example, the species that is now both prey and predator may need a readjusted initial population size in the composed model.

We create three composed models to evaluate our approach, one for each type of composition. An example of the same prey composition, in which both wolves and foxes hunt rabbits, is shown in Figure 3. An example of the food chain composition, in which wolves hunt foxes and foxes hunt rabbits, is shown in Figure 4. Additionally, we created a same predator composition, where the composed model contains one predator that feeds on two prey, i.e., wolves hunt both sheep and rabbits.

2.4 Generating Experiments for Hypotheses

As mentioned before, we consider three properties to be checked (see Section 2.2) and three types of composition (see Section 2.3). Two basic two-species Lotka-Volterra models (see Section 2.2) are reused for each composition. For both models, all three hypotheses are checked, each by its corresponding experiment. All of these hypotheses shall be checked again for composed model. To do so, experiments for the composed model will be generated, based on the experiments that were used to check the hypotheses for the reused models.

Each hypothesis consists of two parts, i.e., parameter value ranges and the property that shall hold (see Section 2.2). The parameter value ranges are reflected by the parameter configuration of the corresponding experiment, whereas the property is a formal statement that depends on the names of model variables (e.g., species names, species population and so on). As described before, these names may be changed by

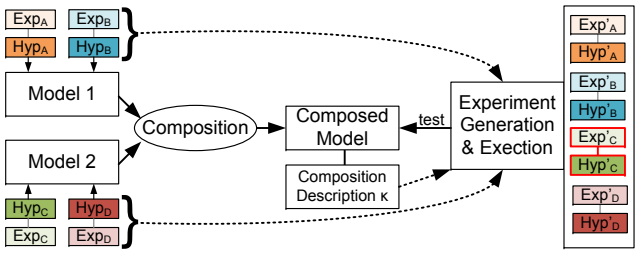

Figure 5: Our overall approach considers the hypotheses of the individual models (colored boxes), as well as the simulation experiments to check them, and supports semantic model composition by first adapting them to the composed model (Hyp'_A etc., right-hand side) and then testing them. The user is notified whenever a hypothesis does not hold for the composed model (e.g., Hyp'_C, see red outline).

the modeler during composition. Therefore, the hypotheses to be checked need to be refined according to those changes.

No species need to be renamed in our example, but the parameter value ranges need reconsideration when generating experiments for the composed model (see Section 3). For the food chain composition, we also adjusted the initial population size of the fox, which is both the prey and the predator in this case. Other details of the experiment setup, such as the simulation stopping criterion, will be copied from the experiments that were defined on the reused models. The hypotheses we consider refer to probabilities, so we follow the approach outlined in Section 2.1.3 to determine the number of necessary replications.

With all this information, we can now execute the newly generated simulation experiments, to check the hypotheses from the reused models on the composed model. We describe our concept and algorithms for this task in the following.

3. THE CONCEPT

The main idea we put forward in this paper is to support semantic composition by automatically adapting and checking hypotheses that are defined on the reused models. To do so, our approach relies on additional information in form of a composition description (see Section 3.2.1). It is then able to identify hypotheses that do not hold in the composed model (or, in case of probabilistic statements, are very unlikely to hold). Figure 5 gives an overview of the general approach.

3.1 Basic Notation

We assume that each model m has an interface, comprising a set of parameters and a set of interaction points [31]. Both are defined by their names and their domain, e.g., the set of parameters $P(m) = \{p_i | i = 1, \ldots, n\}$ with p_i being a tuple $(name, \mathcal{D}_i)$, where $\mathcal{D}_i$ is the domain of the parameter's possible values. Similarly, the set of interaction points of a model is defined, i.e., $IP(m) = \{ip_i | i = 1, \ldots, v\}$ with IP_i being a tuple $(name, \mathcal{D}_i)$. Interaction points are used for aggregating models [29] and for collecting data from the model. Please note that all variables of a model become interaction points when using exchange formats like SBML [17].

Let $\mathbb{A} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ be the set of all permissible parameterizations of the model m. We call an element $a \in \mathbb{A}$ an *assignment*, and the finite subset $A \subseteq \mathbb{A}$ of assignments for which the model has been simulated the *experiment assignments*.

We consider the simulation of a model to be a black box. Given a certain assignment $a \in \mathbb{A}$, and an experiment $exp(a)$, it will generate output in the form of $Y(a)$.

Because simulation can be stochastic, there may be multiple sets of finite trajectories T_i per observation: $Y(a) = \{T_i, i = 1, \ldots, k\}$. Each trajectory set T_i consists of all observed data from a single run, in the form of trajectories tr_i, i.e., $T_i = \{tr_i, i = 1, \ldots, l\}$. We further assume each trajectory tr_i consists of a unique name, e.g., a biological species to be observed, and a sequence of time-stamped data d_i:

$$tr_i = (name, (t_1, d_1), \ldots, (t_z, d_z)).$$

Each trajectory set T_i should also contain corresponding data for each interaction point, i.e.,

$$\forall (name_{ip}, D_{ip}) \in IP(m) : \exists tr_i \in T_i, \\ name = name_{ip} \text{ and } d_j \in D_{ip}\ j \in 1 \ldots z.$$

So far, we have discussed parameter assignments and the output of the experiments in terms of trajectories, which depends on these assignments. Now, we define the properties of a model's behavior. We use a variant of the continuous stochastic logic (CSL) [46, 35] for the stochastic case, which works on multiple replications and where a property expressed in LTL is checked for each replication (see Section 2.1.3). For a certain statement or property $Pr_{\bowtie p}(\phi)_i$, a given assignment set $A_i \subseteq \mathbb{A}$ defines for which parameter assignments $Pr_{\bowtie p}(\phi)_i$ is true. Together, this yields a hypothesis $h_i = \langle A_i, Pr_{\bowtie p}(\phi)_i \rangle$, i.e.,:

$$\forall a \in A_i : \langle a, exp_i(a) \rangle \vDash Pr_{\bowtie p}(\phi)_i. \quad (2)$$

This can be evaluated with statistical model checking, i.e.,

$$\langle a, exp(a) \rangle \vDash Pr_{\bowtie p}(\phi) \iff Prob(\pi \in Path(s) | \pi \vDash \phi) \bowtie p$$

where π represents the T_i, s is the initial state of the model (determined by a and $exp(a)$), and $Y(a) \subseteq Path(s)$. Please note that each hypothesis h_i has a corresponding experiment description exp_i. In the following, we therefore simplify Equation 2 to the shorthand notation $A_i \vDash Pr_{\bowtie p}(\phi)_i$.

The above definition of a hypothesis makes the goal of its corresponding experiment explicit. Statements about the behavior of the model and information about configurations (in terms of parameter assignments) become the focus of interest. The above argumentation neglects information that is essential for reproducing simulation results (like stop criteria, the simulator to use, etc.) which are part of specifying an experiment, $exp(a)$. However, we build on this information when executing the experiments to test the generated hypothesis for the composed model (see Section 2.4). Finally, we assume that each reused model m is annotated with a set of hypotheses $H_m = \{h_i, i = 1, \ldots, q\}$.

3.2 Experiment Generation

3.2.1 Basic Algorithm

We assume that a composed model m_c is created by reusing two models, m_1 and m_2, and that the type of composition is

Algorithm 1 Basic algorithm.
m_c: composed model
H_{m_1}: Hypotheses defined for model m_1
H_{m_2}: Hypotheses defined for model m_2
κ: composition description
u: user preferences

```
1  // Generate possible hypotheses
2  H_mc ← createHypotheses(H_m1, H_m2, κ, u)
3  // Return set for successful hypotheses
4  H+ ← ∅
5  // Return set for failure hypotheses
6  H− ← ∅
7  // Check hypotheses in composed model m_c
8  for each hypothesis h = A_h ⊨ Pr⋈p(φ)_h ∈ H_mc
9    A ← sampleAssignments(h, κ, u)
10   A+ ← ∅
11   A− ← ∅
12   for each assignment a ∈ A
13     exp_a ← generateExperiment(m_c, a, exp_h, Pr⋈p(φ)_h)
14     Y(a) ← run(exp_a)
15     result ← check(Pr⋈p(φ)_h, Y(a))
16     if result is invalid
17       switch(u.returnPreference(h))
18         case strict: return error
19         case record: A− ← A− ∪ a
20                      continue
21       end
22     else
23       A+ ← A+ ∪ a
24     end
25   end for
26   //Add successful hypothesis:
27   H+ ← H+ ∪ {A+ ⊨ Pr⋈p(φ)_h}
28   //Add failure hypothesis:
29   H− ← H− ∪ {A− ⊭ Pr⋈p(φ)_h}
30 end for
31 return H+, H−
```

defined by some $c \in C = \{c_i, i = 1, \ldots, r\}$. In our example, we distinguish the *same prey*, the *same predator*, and the *food chain* composition types (see Section 2.3). Besides the composition type, the modeler may also change the names of some model entities, or change the value ranges of some model parameters. For example, in the food chain composition the initial value of the species population is adjusted during composition (see Section 2.3). These changes, along with the composition type c, are stored in a composition description κ, which describes the overall model composition.

An additional structure for user preferences, u, allows users to influence how hypothesis testing and hypothesis creation are executed. This improves the flexibility of our approach. For instance, some hypotheses must never be violated, e.g., in our example the hypothesis referring to the "empty state" property (prey should always becomes extinct first, followed by the predators, see Section 2.2). A violation of this hypothesis could thus be defined as an error. On the other hand, a violation of the hypothesis referring to the "coexisting state" property (no species becomes extinct, see Section 2.2) may even be *expected* in a composed Lotka-Volterra model, and could thus be tolerated.

Algorithm 1 depicts the procedure of generating experiments for the composed model, based on information about experiments done with the reused models. The input of the algorithm are the composed model m_c, the hypothesis sets H_{m_1}, H_{m_2}, which are defined for the two reused models m_1 and m_2, respectively, the composition description κ, and the user preferences u. First of all, a hypothesis set H_{m_c} of the composed model m_c is created by a function `createHypotheses` (line 2), which will be described in algorithm 2. Additionally, two hypothesis sets to record the result of hypothesis testing, H_+ and H_-, are created (line 4, 6). H_+ stores all hypotheses that hold for the composed model. Similarly, if there are some assignments $a \in A_-$ for which the statement $Pr_{\bowtie p}(\phi)_h$ is not true, this will be stored in H_- as $A_- \not\models Pr_{\bowtie p}(\phi)_i$.

For each hypothesis $h \in H_{m_c}$, experiment assignments A are generated through sampling. This is implemented in a function `sampleAssignments` (line 9), which takes the given hypothesis h, the composition description κ, and the user preferences u as input. While we currently sample uniformly from the assignment set A_h of the given hypothesis, other sampling methods may yield much better results. For example, one could draw samples from nearly orthogonal latin hypercubes (e.g., [33]), or focus on the boundaries of A_h in the parameter space. Since larger sample sizes are computationally more expensive, we expect that the quality of the sampling method will have a large impact on the overall performance. Therefore, users should also be able to adjust the sampling procedure to their requirements, e.g., to generate more samples if the available hardware is sufficiently powerful. This can be expressed via the user preferences u. Similarly, more sophisticated sampling methods may require additional information on the actual model composition, e.g., to generate more samples for parameters shared by both model components.

The sampling results in a set A of experiment assignments. For each assignment a, the algorithm now generates a suitable experiment exp_a. For this, it relies on the composed model m_c, the assignment a, the experiment exp_h that is associated with the hypothesis h, and the statement $Pr_{\bowtie p}(\phi)_h$ of the hypothesis h (line 8), all of which are passed to the function `generateExperiment` (line 13). Currently, this function takes an experiment specification defined in SESSL and adapts it to the composed model and the new assignment. Internally, this function retrieves the experiment definition that corresponds to $Pr_{\bowtie p}(\phi)_h$, which we assume to be preserved from experiments with the reused models m_1 and m_2 (see Section 3.2.2 for details). For example, these definitions can be specified with SESSL, as shown in Figure 2. Apart from the model to be simulated, which is now m_c, the parameters to be used, which are now defined by a, and the number of required replications, which now depends on α, β, and δ (see Section 2.1.3), every aspect of the experiment can stay the same. Thus, this function simply reconfigures the corresponding experiment to work with m_c and a.

With all the details configured, the simulation experiment exp_a can now be executed by invoking `run` (line 14). The output of the simulation execution is a trajectories set $Y(a)$. Then, the statement $Pr_{\bowtie p}(\phi)_h$ will be checked against $Y(a)$. If the check is successful, the assignment a will be stored in the set A_+, which contains all assignments where the current hypothesis h has been corroborated. If the check is unsuccessful, i.e., the result is invalid (lines 16–22), what happens next depends on the user preferences u. If the statement $Pr_{\bowtie p}(\phi)_h$ does not hold for the assignment a, a user may want to let the whole procedure fail as early as possible, because this indicates problems in the model composition or the reused models. On the other hand, it may also be reasonable to check all hypotheses first, and to record which

Algorithm 2 Creation of hypotheses.
H_{m_1}: Hypotheses defined for model m_1
H_{m_2}: Hypotheses defined for model m_2
κ: composition description
u: user preferences

```
1  function createHypotheses(H_m1, H_m2, κ, u)
2    H_mc ← ∅
3    // Create new hypotheses on composed model m_c
4    for each hypothesis h = A_h ⊨ Pr⋈p(ϕ)_h ∈ (H_m1 ∪ H_m2)
5      Pr⋈p(ϕ)'_h ← renameVariables(Pr⋈p(ϕ)_h, κ_names)
6      A_h' ← updateBounds(A_h, κ_A, u)
7      if (A_h' = ∅)
8        switch(u.strictness)
9          case strict: return error
10         case tolerant: continue
11         case retry: A_h' ← κ_A
12       end
13     end
14   H_mc ← H_mc ∪ {A_h' ⊨ Pr⋈p(ϕ)'_h} //Add new hypothesis
15   end
16 return H_mc
```

hold and which do not hold, because this could simplify the later analysis of potential causes. Both approaches are supported by defining $u.returnPreference(h)$ either as `strict` (line 18), which fails early, or as `record` (line 19), which continues the experiments.

3.2.2 Creation of Hypotheses

The creation of hypotheses is described in Algorithm 2. This function works on the hypotheses sets of the two reused models, H_{m_1} and H_{m_2}, and also needs the composition description κ and the user preferences u as input. It iterates over all hypotheses in the two hypothesis sets H_{m_1} and H_{m_2} (line 4). For each hypothesis $h = A_h \vDash Pr_{\bowtie p}(\phi)_h$, the for-loop generates a new statement $Pr_{\bowtie p}(\phi)'_h$ and new experiment assignments $A_{h'}$. The new statement $Pr_{\bowtie p}(\phi)'_h$ is generated by renaming variables, i.e., it depends on the composition description κ (line 5, see Section 2.4). If a variable name has been changed, e.g., substituted with another name in the composed model, this is recorded in κ_{names}. Therefore, we apply all name changes given in κ_{names} to the corresponding variables in statement $Pr_{\bowtie p}(\phi)_h$, which then forms the new statement $Pr_{\bowtie p}(\phi)_h$. This is handled by the function `renameVariables` in Algorithm 2 (line 5).

After model composition, the experiment assignments of the reused models and the experiment assignments of the composed model will usually have different dimensions. One reason for this is that parameters of *both* reused models are included in the composed model. Thus, the experiment assignments of each original hypothesis, A_h, need to be refined to suit the experiment assignment dimensions of the composed model. This is done in a function called `updateBounds`, which takes the assignments A_h, the experiment assignments of the composed model, κ_A, and the user preferences u as input (line 6).

The dimensions of the composed model's experiment assignments, κ_A, are part of the composition description and assumed to be defined by the modeler during composition. This can also be done implicitly. For example, our prototype currently calculates the minimum and maximum value for each parameter of the reused models, by iterating over the experiment assignments of all their hypotheses. To construct κ_A, we use these minimal and maximal parameter values as boundaries, and then join these parameter intervals of the reused models together. If a parameter is defined in both reused models, we use the overall minimum and maximum value. In this way, we can define valid parameter ranges for the composed model without user intervention. Still, if the modeler explicitly sets one parameter to a constant value during the composition, κ_A will be reduced accordingly.

Our prototype currently implements `updateBounds` in a simple way. For each dimension of κ_A, the intersection of the corresponding intervals in A_h and κ_A is calculated. If there is no such dimension in A_h, e.g., because the corresponding parameter is defined in the composed model but not the reused model, the interval from κ_A is used. There are other ways to generate the new experiment assignments $A_{h'}$, e.g., in some situations it may be preferable to compute a union of the parameter intervals (and not an intersection). To let the user have control over this aspect, the function also takes user preferences as input.

Even with these precautions regarding an update of the parameter bounds, it is still possible that A'_h is empty. In our prototype implementation of `updateBounds`, this means there is at least one parameter with non-overlapping bounds in A_h and κ_A. In this case, the user preference regarding the *strictness* of hypothesis generation determines how to proceed. This is shown in lines 8–12 of Algorithm 2. As κ_A describes for which parameter bounds the composed model shall be used, a lack of assignments that fall within these bounds could mean that the composition itself is problematic. Thus, the hypothesis creation function should stop and return an error message (`strict` mode, line 9). However, it could also mean that over-constrained hypotheses are the problem, i.e., they are of less importance for the composed model and should be discarded (`tolerant` mode, line 10). Besides these options, a user might also be more interested in exploring the validity of model properties for the new parameter bounds of the composed model, and would thus like to check them with the experiment assignments in κ_A (`retry` mode, line 11). Anyhow, the newly defined experiment assignments $A_{h'}$ will then be used to define a new hypothesis (line 14).

We assume that for each hypothesis $h = A_h \vDash Pr_{\bowtie p}(\phi)_h$, defined for either m_1 or m_2, there is also a corresponding experiment definition exp_h that allows to check whether $Pr_{\bowtie p}(\phi)_h$ is true for a certain assignment $a \in A_h$ and the reused model. As described in Section 3.2.1, this experiment definition can be reused with minimal adaptations. Thus, it allows us to check whether $Pr_{\bowtie p}(\phi)'_h$ is true for a certain assignment $a' \in A'_h$ and m_c. The advantage of this approach is that the actual experiment definition can be arbitrarily complex, e.g., in terms of output analysis, experiment design, or stopping conditions. Anyhow, we omit the explicit handling of experiments from the pseudo-code for simplicity.

3.2.3 Interpretation of Generated Output

If Algorithm 1 finds a problem, it will return a (set of) counterexamples H_{-}. It is easy to re-check these counterexamples on the model for which they have been defined originally. This is important, because the hypotheses that characterize the reused models may have been checked insufficiently, i.e., they may not even be valid for the original models. Depending on the outcome of testing the corresponding model component (m_1 or m_2), the user knows whether there

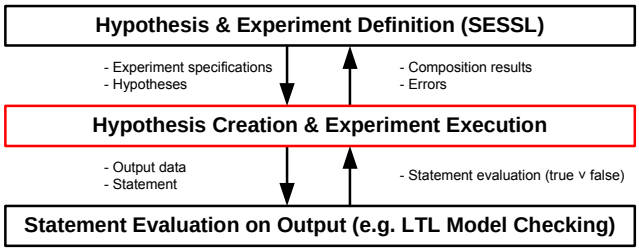

Figure 6: The three layers of our prototype implementation: a SESSL extension is used to specify experiments and hypotheses on model components (upper layer) and the middle layer generates the new hypotheses and executes all experiments (red box). The lower layer evaluates each statement against the generated output.

is a problem with the composition (i.e., the counterexample is true for m_1, but not for m_c) or a problem with the hypothesis set of the original model component (i.e., the counterexample is true for neither m_1 nor m_c). Such checks can be easily triggered automatically, so that a user can quickly see what kind of problem has been encountered. Extending our prototype in that regard will be subject to future work.

4. IMPLEMENTATION & RESULTS

4.1 Implementation

Our implementation of the concepts described in Section 3 consists of three distinct layers, as summarized in Figure 6.

The first layer provides the user interface, i.e., it allows to define hypotheses and their corresponding simulation experiments for individual models. Currently, this user interface is realized as an extension of SESSL [9] that provides the functionality to define hypotheses regarding the simulation output. It can be used to augment SESSL experiments by mixing in the `Hypotheses` trait, as shown in line 1 of Figure 2. Besides defining simulation experiments, the main task of this layer is the creation of an object hierarchy that represents a hypothesis. Both the experiment specification and the hypothesis representation are handed over to the second layer. Many other kinds of user interfaces could be developed on top of this layer, e.g., to specify hypotheses via a graphical user interface.

The second layer implements our overall concept (see Algorithms 1 and 2) and defines the type hierarchy for the hypotheses that are currently supported, i.e., so far LTL-Expressions, probabilistic statements ($Pr_{\bowtie p}(\phi)$), and custom predefined predicates (see Section 2.1.3). This layer is integrated into JAMES II, so that it can leverage its plug'n simulate approach [16] to ensure its flexibility regarding future extensions, as well as its applicability to the various modeling formalisms already supported by JAMES II.

The third layer is triggered by the second layer to check the generated hypotheses against the simulation output of the composed model. It is implemented on top of JAMES II as well. So far, this layer provides a re-implementation of the model checking approach followed in [10], as well as support for custom predicates. We consider the integration of additional model checking approaches to be future work.

	Same Prey	Same Predator	Food Chain
Coexisting state	√ / √	⊘ / ⊘	√ / ⊘
Empty state	√ / √	√ / √	⊘ / √
Recovery comparison	√ / √	⊘ / √	√ / ⊘

Table 1: Results overview. For each composition type, the hypotheses of the two reused models are checked. Each cell contains the results of checking both hypotheses that refer to the same property (one hypothesis from each of the reused models). The symbol √ represents hypotheses that hold, whereas the symbol ⊘ represents those that do not hold.

Finally, note that our prototypical implementation is still mostly independent of concrete simulation systems, i.e., it can be integrated into other simulation systems with relatively little effort. This is because the plug-in system of JAMES II is used as a foundation, but our prototype requires none of its simulation-specific abstractions.

4.2 Result Analysis

Table 1 gives an overview about the results of our example (see Section 2), where √ means that the property also hold in the composed model, and ⊘ that it does not hold (at least not with the expected probability $p \geq 0.8$). For each type of composition (see Section 2.3), the hypotheses referring to the two reused models are checked for the composed model.

If we look at the results, we see that two predators feeding on the same prey is unproblematic (i.e., the *same prey* composition). This is to be expected, as the prey still controls the behavior of the predator(s). The situation is different if a predator feeds on two prey species (i.e., the *same predator* composition). Here, the weaker prey is likely to become extinct, which is not uncommon. In case of the *food chain* composition, only the hypothesis of one reused model holds in the composed model, while the hypothesis of the other reused model does not hold. This indicates that this composition type might not be valid. Actually, in the given example we would expect the wolves to also feed on rabbits as soon as rabbits are around, and not only on foxes (cf. Figure 4).

Table 2 shows the results in more detail. Here, another interesting observation is that one hypothesis regarding "Recovery comparison" for the *same predator* composition holds for 9 out of 10 tested assignments in the composed model. Thus, the likelihood to find a counterexample is low, i.e., this would be difficult to find out if the hypotheses were checked manually. This illustrates the potential of our approach in preventing users from coming to incorrect conclusions regarding the hypotheses that hold in a composed model.

5. DISCUSSION

5.1 Related work

Other approaches that address semantic composition of models depend on the existence of a perfect model, based on which the composed models can be validated [44, 38]. The validity of the composed model is measured by comparing its simulation executions with that of the perfect model, which are represented by Labeled Transition Systems [37].

Original statement	Composition Type	Composition Description	$\lvert A_+\rvert$	$\lvert A_-\rvert$	Counterexample
Coexisting State(Rabbit, Fox)	The Same Prey	–	10	0	–
Coexisting State(Rabbit, Wolf)	The Same Prey	–	10	0	–
Coexisting State(Rabbit, Wolf)	The Same Predator	–	3	7	a:0.015;b:0.7;d:0.5; nRabbit:100;nWolf:10;nSheep:100
Coexisting State(Sheep, Wolf)	The Same Predator	–	1	9	a:0.012;b:0.9;d:0.5; nRabbit:100;nWolf:10;nSheep:100
Coexisting State(Rabbit, Fox)	The Food Chain	nFox:30	10	0	–
Coexisting State(Fox, Wolf)	The Food Chain	nFox:30	5	5	a:0.014;b:0.7;d:0.9; nRabbit:100;nWolf:10;nFox:30
Recovery Comparison(Rabbit, Fox)	The Same Prey	–	10	0	–
Recovery Comparison(Rabbit, Wolf)	The Same Prey	–	10	0	–
Recovery Comparison(Rabbit, Wolf)	The Same Predator	–	9	1	a:0.024;b:0.9;d:0.5; nRabbit:10;nWolf:10;nSheep:10
Recovery Comparison(Sheep, Wolf)	The Same Predator	–	10	0	–
Recovery Comparison(Rabbit, Fox)	The Food Chain	–	10	0	–
Recovery Comparison(Fox, Wolf)	The Food Chain	–	10	0	–
Empty State(Rabbit, Fox)	The Same Prey	–	10	0	–
Empty State(Rabbit, Wolf)	The Same Prey	–	10	0	–
Empty State(Rabbit, Wolf)	The Same Predator	–	10	0	–
Empty State(Sheep, Wolf)	The Same Predator	–	10	0	–
Empty State(Rabbit, Fox)	The Food Chain	nFox:30	0	10	a:0.055;b:0.3;d:0.8; nRabbit:100;nWolf:30;nFox:30
Empty State(Fox, Wolf)	The Food Chain	nFox:30	10	0	–

Table 2: Detailed results. The column "Original statement" shows the properties and the related species. All properties are checked regarding their probability ($p \geq 0.8$), which is omitted for simplicity. In "Composition Description", additional information regarding the composition are given (if necessary). $\lvert A_+\rvert$ and $\lvert A_-\rvert$ are the number of assignments for which the property holds and does not hold, respectively. In our example, we set the sampling number to 10, thus $\lvert A_+\rvert + \lvert A_-\rvert = 10$. A counterexample is given for invalidated hypotheses (see rows with gray background).

However, it may not be easy to find a perfect model. In contrast, our approach makes use of models being annotated with experiments and hypotheses about a model's behavior. Through checking the hypotheses of the reused models in the composed model, some information about the validity of the composed model are provided. Hence, in our approach, the semantic model composition is studied from a different view.

With the initiative MIASE (Minimum Information About a Simulation Experiment) [42], and associated methods like SED-ML (a description language for experiments) [43], the systems biology community set out to define standards to annotate models with experiment descriptions. While these annotations are typically aimed at reproducing simulation results, we use this information about experiments to derive hypotheses that ensure semantically more reasonable composition, and for testing hypotheses by adapting the original experiment specifications. Thereby, we move the attention from describing the details needed to reproduce or execute an experiment to the goal that drives an experiment, i.e., the definition of the hypothesis to be tested.

Increasingly, trajectories produced by simulation are checked regarding specific properties, often defined in temporal logic. Some approaches work with deterministic models [10], while others work with stochastic models [35]. This is reflected in the language that is used to express the properties that shall hold. Whereas linear temporal logic (LTL) is focused on one trajectory, continuous stochastic logic (CSL) has been developed to check Continuous Time Markov models. In our approach we rely on a hybrid approach, i.e., we use LTL to check individual traces and adopt a CSL construct as an external wrapper to express the expected probability. However, currently the nesting of probabilistic operators is not supported. To evaluate the LTL formulas, we use the algorithm presented in [10], and to account for the stochasticity we adopt the approach from [35] (see Section 2.1.3).

As properties defined in temporal logic can be interpreted as a discrete target function, this approach can easily be applied for optimizing the model, e.g., to fit the parameter values of a model, including the parameters of rules that link two or more models [20]. Those approaches depend on users specifying explicitly the entire experiment from scratch, including the goal of the individual experiment in terms of the parameter value ranges to be searched and the property to be checked. In our approach, we extract this information by reusing "old" experiment descriptions, i.e., we interpret an experiment description as a hypothesis that holds for a reused model (and experiment, e.g., regarding parameter assignments), and can thus be used to automatically generate and check this hypothesis for the composed model. Thus, our approach takes a next step to generate hypotheses and experiments automatically, by reusing hypotheses and experiment descriptions of the models that shall be reused.

Techniques similar to ours have also been developed in the field of software testing. For example, program analysis via symbolic execution can be combined with model-checking and observations from actual program executions to automatically generate software tests [28]. These tests can be considered as experiments to test developer hypotheses regarding a program, e.g., that it should not crash for valid input. Test generation may even exploit the composition of a program, which in this context means to construct the overall set of execution paths by considering the execution paths of each procedure individually [13]. Like the sampling of experiment assignments (see Section 3.2.1), the genera-

tion of test data is crucial to find bugs, i.e., to invalidate a hypothesis. Metaheuristics have been successfully applied to this task [23], and we plan to integrate similar techniques to our prototype.

5.2 Limitations and future work

A limitation of our approach is that the falsification or corroboration of hypotheses referring to the composed model and certain assignments still have to be interpreted by the user. There does not seem to be an easy way to automate this step. However, the approach provides additional valuable information to the user for evaluating the composed model. A suitable presentation of this information still has to be developed.

The current language to describe properties (see Section 2.1.3) is not able to easily express all interesting aspects even in our comparatively simple experiments with the Lotka-Volterra models. Therefore, future work will be dedicated to enhance the expressiveness of these LTL terms, e.g., integrating regular expressions to describe repeating patterns [5] (e.g., oscillations), or integrating user-defined functions. Both will be facilitated by the design of SESSL as a domain-specific language embedded in Scala.

Our current approach only considers individual trajectories and replications of those. However, some properties refer to differences between the trajectories sets of multiple assignments. For example, the ratio-dependent theory states that if an ecosystem has richer resources, there should be higher equilibrium abundances on all trophic levels, in comparison to an ecosystem with less resources [12]. So, if we have more food for rabbits available, we would expect a higher number of both, rabbits and foxes. To also allow for these kind of properties to be checked, the current design of Algorithm 1 has to be adapted, as it requires to compare the relation between the results of different assignments.

Compared to the original hypotheses, the hypotheses that are currently checked on the composed model are only different referring to possibly renamed species and the sampling of parameter values for which the statements shall hold. Depending on the number of hypotheses to be checked, some preprocessing could allow a more efficient evaluation of LTL formulas, e.g., by joining some formulas together or by reordering their operators, and should thus be considered in the future. In addition, as already mentioned in Section 3.2, sampling methods may have a strong impact on overall performance and need to be investigated further.

6. CONCLUSION

An explicit, unambiguous specification of experiments is increasingly required to support the reproduction of simulation results in many application domains. This information can also be very valuable to support a semantically meaningful reuse of models. The underlying idea of our approach is that, if models are annotated with experiments and their corresponding hypotheses, then it becomes possible to automatically generate new experiments and hypotheses for composed models, and to automatically execute the new experiments to check the hypotheses, independently of what kind of composition, e.g., aggregation or fusion, is used.

Obviously, the results of these experiments help assessing the validity of the newly composed model, referring to the questions the reused models have been designed for.

Our approach has been realized in the context of the modeling and simulation framework JAMES II. For experiment specification and execution, we use the embedded domain-specific language SESSL, which we extended to allow the definition of experiment hypotheses. The hypotheses comprise statements about the behavior of the system and parameter ranges within which this behavior has been observed. As we are working with stochastic models, the statements shall hold with a specific probability. To do so, statements of the form $Pr_{\bowtie p}(\phi)$ are defined by adopting concepts from continuous stochastic logic and linear temporal logic (see Section 5.1), where ϕ describes a property of an individual trajectory (and thus is tested on individual trajectories), whereas assessing the probability p with which ϕ can be observed depends on statistical model checking methods, i.e., it is based on simulation replication.

After our promising results with the Lotka-Volterra model, we plan to apply our approach to current modeling and simulation studies in the area of cell biology, where a Wnt-pathway model [22] is successively extended by other models that describe membrane-related dynamics or diffusion processes in more detail, and which come with their own experiments and hypotheses. The results will shed new light on the validity of the newly developed models, or, in some cases, also on the validity of hypotheses assumed to hold for the reused models. We expect our approach to significantly improve the development process of these new models, as it helps to ensure a semantically meaningful reuse of the existing models.

Acknowledgments

This research is partly supported by the CSC (China Scholarship Council), the German research foundation, (Grant No. EW 127/1-1), and the National Natural Science Foundation of China (Grant No. 61374185). We thank Tom Warnke for providing his implementation of the algorithm presented in [10].

7. REFERENCES

[1] J. Bézivin, S. Bouzitouna, M. D. Del Fabro, et al. A canonical scheme for model composition. In *Model Driven Architecture–Foundations and Applications*, pages 346–360. Springer, 2006.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[3] P. K. Davis and R. H. Anderson. Improving the composability of DoD models and simulations. *JDMS*, 1(1):5–17, Apr. 2004.

[4] L. De Alfaro and T. A. Henzinger. Interface-based design. In *Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, M. Broy, J. Gruenbauer, D. Harel, and C.A.R. Hoare, 2005.

[5] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 854–860. AAAI Press, 2013.

[6] M. Droz and A. Pękalski. Different strategies of evolution in a predator-prey system. *Physica A: Statistical Mechanics and its Applications*, 298:545–552, 2001.

[7] P. Du, W. A. Kibbe, and S. M. Lin. Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics*, 22(17):2059–2065, 2006.

[8] H. Elmqvist, S. E. Mattsson, and M. Otter. Object-oriented and hybrid modeling in modelica. *Journal Européen des systèmes automatisés*, 35(1):1–10, 2001.

[9] R. Ewald and A. M. Uhrmacher. SESSL: A Domain-Specific Language for Simulation Experiments. *ACM Transactions on Modeling and Computer Simulation*, 2014 (to appear). See http://sessl.org.

[10] F. Fages and A. Rizk. On the analysis of numerical data time series in temporal logic. In *Computational Methods in Systems Biology*, 2007.

[11] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, 81(25), 1977.

[12] L. Ginzburg and H. Akçakaya. Consequences of ratio-dependent predation for steady-state properties of ecosystems. *Ecology*, 73(5):1536–1543, 1992.

[13] P. Godefroid. Compositional dynamic test generation. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 47–54, New York, NY, USA, 2007. ACM.

[14] A. Hallagan, B. Ward, and L. F. Perrone. An experiment automation framework for ns-3. In *Proceedings of the 3rd Int'l ICST Conference on Simulation Tools and Techniques*. ICST, 2010.

[15] T. Helms, M. Luboschik, H. Schumann, and A. M. Uhrmacher. An approximate execution of rule-based multi-level models. In *Proceedings of the 11th International Conference on Computational Methods in Systems Biology*, 2013.

[16] J. Himmelspach and A. M. Uhrmacher. Plug'n simulate. In *Proceedings of the 40th Annual Simulation Symposium*, ANSS '07, pages 137–143, Washington, DC, USA, 2007. IEEE Computer Society.

[17] M. Hucka, L. Smith, D. Wilkinson, M. Hucka, et al. The systems biology markup language (SBML): language specification for level 3 version 1 core. *Nature Precedings*, Oct. 2010.

[18] C. Li, M. Donizelli, N. Rodriguez, et al. BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Systems Biology*, 4:92, Jun 2010.

[19] A. Lotka. *Elements of Physical Biology*. Williams & Wilkins Company, 1925.

[20] E. D. Maria, F. Fages, and S. Soliman. On coupling models using model-checking: Effects of irinotecan injections on the mammalian cell cycle. In *Proceedings of Computational Methods in Systems Biology*, pages 142–157. Springer, 2009.

[21] C. Maus, S. Rybacki, and A. M. Uhrmacher. Rule-based multi-level modeling of cell biological systems. *BMC Systems Biology*, 5(166), 2011.

[22] O. Mazemondet, M. John, S. Leye, A. Rolfs, and A. M. Uhrmacher. Elucidating the sources of beta-catenin dynamics in human neural progenitor cells. *Plos One*, 7(8):e42792–e42792, 2012.

[23] P. McMinn. Search-based software test data generation: a survey. *Software Testing, Verification and Reliability*, 14(2):105–156, June 2004.

[24] D. Nicol, C. Priami, H. Nielson, and A. Uhrmacher, editors. *Simulation and Verification of Dynamic Systems*. Dagstuhl Seminar Proceedings 0161, 2006. ISSN 1862-4405.

[25] C. M. Overstreet, R. Nance, and O. Balci. Issues in Enhancing Model Reuse. In *First International Conference on Grand Challenges for Modeling and Simulation*, 2002.

[26] D. Peng, A. Steiniger, T. Helms, and A. Uhrmacher. Towards Composing ML-Rules Models. In *Proceedings of the 2013 Winter Simulation Conference*, 2013.

[27] M. D. Petty and E. W. Weisel. A composability lexicon. In *Spring Simulation Interoperability Workshop (SISO)*, pages 181–187, 2003.

[28] C. Păsăreanu and W. Visser. A survey of new trends in symbolic execution for software testing and analysis. *International Journal on Software Tools for Technology Transfer*, 11(4):339–353, Oct. 2009.

[29] R. Randhawa, C. a. Shaffer, and J. J. Tyson. Model aggregation: a building-block approach to creating large macromolecular regulatory networks. *Bioinformatics (Oxford, England)*, 25(24):3289–95, Dec. 2009.

[30] E. Renshaw. *Modelling Biological Populations in Space and Time*. Cambridge Studies in Mathematical Biology. Cambridge University Press, 1991.

[31] M. Röhl and A. M. Uhrmacher. Definition and analysis of composition structures for discrete-event models. In *Proceedings of the 2008 Winter Simulation Conference*, pages 942–950, 2008.

[32] S. Rybacki, F. Haack, K. Wolf, and A. Uhrmacher. Developing simulation models - from conceptual to executable model and back - an artifact-based workflow approach. In *SimuTools*, 2014.

[33] S. Sanchez and H. Wan. Work smarter, not harder: A tutorial on designing and conducting simulation experiments. In *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pages 1–15, Dec 2012.

[34] F. Scholkmann, J. Boss, and M. Wolf. An efficient algorithm for automatic peak detection in noisy periodic and quasi-periodic signals. *Algorithms*, 5(4):588–603, 2012.

[35] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In K. Etessami and S. Rajamani, editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 266–280. Springer Berlin Heidelberg, 2005.

[36] C. A. Shaffer, R. Randhawa, and J. J. Tyson. The role of composition and aggregation in modeling macromolecular regulatory networks. In *Proceedings of the 38th conference on Winter simulation*, pages 1628–1636. Winter Simulation Conference, 2006.

[37] J. Srba. On the power of labels in transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory*, pages 277–291, 2001.

[38] C. Szabo and Y. Teo. An approach for validation of semantic composability in simulation models. *... on*

Principles of Advanced and Distributed Simulation, pages 3–10, June 2009.

[39] C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-oriented Programming*. ACM Press Series. ACM Press, 2002.

[40] A. Tolk. What comes after the semantic web - pads implications for the dynamic web. In *Workshop on Principles of Advanced and Distributed Simulation (PADS)*, page 55. IEEE Computer Society, 2006.

[41] V. Vito. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Mem. R. Accad. Naz. dei Lincei*, 2:31–113, 1926.

[42] D. Waltemath, R. Adams, D. A. Beard, F. T. Bergmann, et al. Minimum Information About a Simulation Experiment (MIASE). *PLoS Computational Biology*, 7(4):e1001122, 2011.

[43] D. Waltemath, R. Adams, F. Bergmann, M. Hucka, et al. Reproducible computational biology experiments with SED-ML - the simulation experiment description markup language. *BMC Systems Biology*, 5:198, 2011.

[44] E. Weisel, M. Petty, and R. Mielke. Validity of models and classes of models in semantic composability. *Proceedings of the Fall 2003 SIW*, 2003.

[45] H. L. Younes and R. G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368 – 1409, 2006.

[46] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings 14th International Conference on Computer Aided Verification, volume 2404 of LNCS*, pages 223–235. Springer, 2002.

[47] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 243–252, New York, NY, USA, 2010. ACM.