# Reusing simulation experiment specifications to support developing models by successive extension

Danhua Peng[a,*], Tom Warnke[a], Fiete Haack[a,b], Adelinde M.Uhrmacher[a]

[a] *Institute of Computer Science, University of Rostock, Rostock 18059, Germany*
[b] *Research Unit "Functional Genome Analysis", Leibniz Institute for Farm Animal Biology (FBN), Dummerstorf 18196, Germany*

**Abstract**

Model development is a successive process of validating, revising, and extending models, and requires iterative execution of simulation experiments. While developing a model by extension, executing similar simulation experiments to those performed with the original model reveals important behavioral insights into the extended model. An automatic generation and execution of these simulation experiments can provide valuable support in the process of developing models. A prerequisite is an explicit specification of simulation experiments. Therefore, we annotate models with simulation experiments that are specified in a declarative domain specific language SESSL (Simulation Experiment Specification via a Scala Layer). Based on experiment specifications of the original model, we introduce a mechanism to automatically generate and execute simulation experiments for the extended model with necessary adaptations. Furthermore, as we experiment with stochastic models, we exploit statistical model checking and specify the expected model behavioral properties, against which the simulation results are checked. Thereby, when a model is extended, the original experiment specifications are reused, adapted, and applied to the extended model. Accordingly, the generated simulation trajectories are probed to check whether the expected properties hold with a certain probability or not. Thus, more fast and frequent feedback during model development can be provided to the modeler. Based on a model of membrane related dynamics, we show how the developed approach can be used in successively extending models.

*Keywords:*
Model extension, Stochastic modeling, Simulation experiments, Experiment specification, Experiment generation and execution

## 1. Introduction

Developing models presents itself as an intricate process and requires iterative experimentation, validation and refinement, as described by different life-cycles, e.g., in [1].

*Corresponding author
*Email addresses:* danhua.peng2@uni-rostock.de (Danhua Peng), tom.warnke@uni-rostock.de (Tom Warnke), fiete.haack@uni-rostock.de (Fiete Haack), lin@informatik.uni-rostock.de (Adelinde M.Uhrmacher)

Often, models are generated incrementally. As soon as one part of the model satisfies certain requirements (after iteratively executing experiments and revising the model), new parts can be added. When a model is extended, it would be interesting to know whether the extended model still behaves the same way as before, and which "properties" of its behavior remain unchanged and which do not. Simulation experiments can provide key insights into model behavior. Therefore, to analyze the impact that the extension has on the model behavior, similar simulation experiments to those that have been conducted with the original model are conducted with the extended model.

However, re-conducting prior simulation experiments on the extended model, especially by hand, is non-trivial and error-prone. First of all, the information of each simulation experiment conducted on the original model needs to be recorded, along with corresponding simulation results, for example in document files or spreadsheets. Based on this information, simulation experiments are manually designed and executed for the extended model. For each experiment configuration, the results from different model versions are analyzed and compared. The situation is aggravated when models are successively extended and many simulation experiments are conducted for each model version during this process. Therefore, in this paper, we present an approach to support the automatic generation and execution of experiments upon the extended model by reusing the information about simulation experiments of the original model. Our goal is to provide fast and frequent feedback during model development to the modeler.

Our work is based on explicit, unambiguous experiment specifications for executing simulation experiments. To interweave modeling and experimentation, models are annotated with specifications of simulation experiments that have been executed, so that those experiment specifications can be reused in revising and extending models. At the moment a model is extended, the original simulation specifications are reused, adapted, and applied to the extended model. As we focus on stochastic models and to make the properties to be checked explicit, we make use of well-established statistical model checking techniques to analyze simulation results, i.e., formalizing the interesting model behavior properties and checking simulation results against those properties.

Even though the individual aspects of our approach, i.e., explicit specification of simulation experiments (e.g., see [2]) and statistical model checking technique (e.g., see [3]), are not new, the contribution of our approach lies in how they are brought together to provide assistance in the process of model development by successively revising, extending, and validating models.

In the following, we first discuss different aspects in reusing simulation experiments, i.e., specifying simulation experiments, specifying properties of model behavior and specifying the data processing method. Next, we present the method for checking properties. Afterwards, we illustrate the automatic experiment generation for the extended model based on experiment specifications of the original model. For demonstration, we applied our approach to the development of receptor dynamics models. Finally, we discuss related work and the usability of our approach, and summarize the results.

## 2. Requirements for reusing simulation experiments

The essential idea of our approach is to reuse simulation experiments. To facilitate the reuse of simulation experiments, an explicit, unambiguous description of experiments is needed. For that, the information required has to be identified.

### 2.1. Structuring information about simulation experiments

An unambiguous and complete description of simulation experiments is of crucial importance for their reproducibility and reuse. Many efforts have been dedicated to define standards for describing simulation experiments, e.g., Minimum Information About a Simulation Experiment (MIASE) [4] and Minimum Simulation Reporting Requirements (MSRR) [5]. Following those guidelines, we distinguish four important aspects for specifying simulation experiments: model configuration, simulation configuration, data processing method and model behavioral property.

*Model configuration* defines how the model is used in the simulation experiment, including its location, its initial state, and configuration of model parameters. *Simulation configuration* describes the set-up of the simulation experiment (e.g., simulation algorithm and stopping rules). *Data processing method* describes the method used to process the output data of the simulation experiment, such as a smoothing method. *Model behavioral property* defines the requirements a model's behavior has to satisfy, which are reflected in the generated experiment output. For example, in a cell model, one property could be that the concentration of a certain protein should reach a steady state after a certain simulation time.

### 2.2. Specifying simulation experiments

Numerous approaches exist to support the specification of experiments. For instance, the Integrated Modeling Support Environment (IMSE) Experimenter, which is a graphical tool to support experimentation with performance models [6], and the Experiment Schema Extension (Ex-SE) of a framework [7], which combines performance model interchange formats and experiment specifications to execute and analyze performance experiments, are developed to facilitate the description, execution, and documentation of experiments based on formal models. In addition, some other work exploits general scientific workflow systems, such as Taverna [8], to specify simulation experiments, e.g., [9].

In the last decade, a series of *domain specific languages* (DSL) have been developed to express different aspects of a simulation experiment [10]. A DSL is a language specialized to a particular problem that "speaks" the language of the domain [11].

The Simulation Experiment Description Markup Language (SED-ML) [12] is an XML-based language to encode and document simulation experiment information required by MIASE, to facilitate exchange and reproduction of experiments in systems biology. It allows describing most frequent types of simulation experiments and is independent of concrete simulation systems. However, it only supports models described in XML and encodes the description of simulation experiments in XML, which makes it more machine-readable than human-readable, and therefore requires additional tools (e.g., SED-ED) to be used by modelers [13]. Similarly, the ns-3 Experiment Description Language (NEDL) [14] and the SAFE Language for Experiment Description (SLED) [10] are two external domain specific languages for experiment description as well. While NEDL is based on XML and SLED is based on JSON, both of them are specific to network simulation.

The Simulation Experiment Specification via a Scala Layer (SESSL) is an internal domain specific language for simulation experiments [2]. It supports specifying experiments in a declarative style for different simulation systems. Through creating a binding to a specific simulation system in SESSL, the simulation experiments specified with SESSL are actually performed with this simulation system. SESSL uses syntactic constructs

of its host language Scala to create the "feel" of a simulation specification language. By allowing the user to set up and to conduct simulation experiments regardless of the software that actually executes the simulations, it represents an extra layer between the user and the simulation system. The set-up of simulation experiments in SESSL consists of not only the model configuration, but also the simulation configuration. Furthermore, SESSL provides different bindings for analysis, e.g., simulation-based optimization.

To make it easy to follow, we explain the basics of SESSL with a concrete example. A simple example is shown in Figure 1, which specifies and executes a simulation experiment for the modeling and simulation framework JAMES II [15] via importing the binding denoted by `sessl.james._` (line 2). Other simulation systems can be used by replacing the binding in the second line. As shown in line 3, an experiment is defined by instantiating an anonymous sub-class of the class `Experiment`, and different traits can be mixed in when necessary. For instance, the trait `Observation` facilitates the specification for result processing, such as how to collect experiment output, and the trait `ParallelExecution` allows the defined experiment to exploit the parallel resources available. In SESSL, based on the features of the host language Scala, the definition of an experiment is realized by the constructor of the anonymous class (line 4-12) and each line is a function invocation, which may appear like an assignment (e.g., line 4 and 7). Line 4 indicates that a Lotka-Volterra model, formalized in ML-Rules [16], is used for experimentation by assigning the SESSL keyword `model` with the location "`file-mlrj:/./LotkaVolterra.mlrj`". Line 5 specifies the configuration of model parameters `nPredator` and `nPrey`, whose value are set as 50 and 500, respectively. Apart from parameter configuration with fixed values, parameter scan is supported as well, as shown in line 6, where the model parameter `nFood` is iterated over the range 100 to 200 with step size 10. The simulation algorithm to use is determined by assigning the SESSL keyword "simulator" with the simulator name, which is a defined case class in SESSL (see [2] for more detail). As depicted in line 7, the simulation algorithm implemented in the case class "MLRulesReference" (presented in [17] and [18]) is exploited for experiments. Line 8 specifies the simulation stop conditions, i.e., the simulation stops when the wall clock time is 1 second or the simulation time is 500. During the simulation, the variable `nPredator` is observed (line 9) and recorded every time unit from time 0 to time 500 (line 10), by including the trait `Observation` (line 3). For each set-up, 10 simulation runs are carried out (line 11) and all except 2 of available cores shall be used for parallel execution (line 12). Line 14 indicates executing the defined experiment.

Besides all the features it provides, including the specification of model configuration and simulation configuration, the most important aspect of SESSL for our approach is that, as an internal domain specific language, it can be easily extended with new features. Therefore, we chose SESSL to specify experiments, and extended it to allow the specification of the data processing method and those required to support automatic property checking.

### *2.3. Specifying the expected behavioral property*

An explicit and unambiguous specification of the model and simulation configuration allows for reproducing simulation experiments. However, the desired or expected behavior of the model observed through the simulations has to be specified as well.

The description of model behavior can be supported by ontologies like TEDDY (TErminology for the Description of DYnamics) [19]. TEDDY provides a machine-

```
import sessl._
import sessl.james._
val exp = new Experiment with Observation with ParallelExecution {
  model = "file-mlrj:/./LotkaVolterra.mlrj"
  set("nPredator" <~ 50, "nPrey" <~ 500)
  scan("nFood" <~ range(100, 10, 200))
  simulator = MLRulesReference()
  stopCondition = AfterWallClockTime(seconds=1) or AfterSimTime(500)
  observe("nPredator")
  observeAt(range(0, 1, 500))
  replications = 10
  parallelThreads = -2
}
execute(exp)
```

Figure 1: A simple SESSL experiment specification.

readable classification of model behavior. However, as TEDDY focuses only on the most critical features of experiment results through providing vocabularies, like increasing or oscillation, this classification will in most cases not be sufficiently precise to capture a certain model behavior and set it apart from irrelevant ones. Another method to annotate models with experimental results is SBRML [20], which associates the model, the process applied to the model and the data resulting from this process based on defining ontologies in XML. While TEDDY aims at providing semantic information of models and therefore facilitating their reuse, SBRML was developed for communicating and exchanging simulation results. In our case we are interested in the process of developing models by successive extension. Accordingly, rather than describing *observed* model behavior as in the approaches above, we are interested in the specification of *expected* model behavior. For example, a recorded experiment result could contain an experiment set-up on a model that exhibits a steady state with a certain value $x$, but executing experiments with the same set-up on the extended model might yield a steady state with a value $x'$. Without further information about the expected behavior, it is impossible to infer whether $x$ and $x'$ are sufficiently close to each other so that the extended model still meets the expectation.

Thus, our approach requires more precise specifications, possibly combining qualitative and quantitative descriptions. This applies particularly to the representation of temporal characteristics, as these play a central role in the description of dynamic model behavior.

During the last decades, temporal logics and model checking techniques [21] are increasingly used for modeling and analyzing systems, e.g., in [22]. Properties are formalized in temporal logics and then automatically verified with model checking algorithms. To meet different requirements of specifying properties, numerous variants of temporal logics based on LTL (Linear Temporal Logic) and CTL (Computational Tree Logic) have been proposed, e.g., PCTL (Probabilistic Computation Tree Logic) [23], CSL (Continuous Stochastic Logic) [24], QFTL($\mathbb{R}$) (Quantifier-Free First-Order LTL over the Reals) [25], MITL (Metric Interval Temporal Logic) [26], and BLTL (Bounded Linear Temporal Logic) [27]. These temporal logic approaches rely on modal operators to enrich logical formulas with temporal aspects. Temporal properties can be expressed qualitatively without explicit statements about time.

In many domains this can be seen as an advantage as it lowers the complexity of

the expressions. However, this also limits the expressive power and the possibilities to formulate statements with complex temporal characteristics. For example, temporal logic extensions usually include a variant of the "globally" operator $\boldsymbol{G}$. $\boldsymbol{G}_{[0,t]}(inc(v))$ might state that the variable $v$ increases strictly during the next $t$ time units (cf. [28]). While probabilistic variants are able to express that $v$ increases during the interval with a probability of at least $p\%$, it is not possible to express that $v$ increases during at least $p\%$ of the interval. However, such a broad definition of the increase of a variable might be appropriate in a stochastic model. Temporal logics cannot completely capture such properties of stochastic trajectories. They are not able to deal with fluctuations in the simulation trajectories that obscure the relevant properties.

As an alternative to modal operators, temporal characteristics can be incorporated in logical expressions by augmenting first-order logic with temporal arguments [29]. Two approaches can be distinguished: Reified temporal logics exploit truth predicates that connect a non-temporal statement with time points or intervals [30] (e.g., $HOLDS[t_1, t_2, incr(v)]$); Non-reified temporal logics employ predicates with non-temporal and temporal arguments simultaneously [31] (e.g., $incr(t_1, t_2, v)$). In both approaches, predicates on model variables are always evaluated with respect to time points or intervals and thus closely related to the time axis, facilitating precise temporal statements. Model variables, time points, and intervals can then further be related to each other using logical statements. The expressive power of first-order logic allows for the definition of noise-tolerant descriptions. However, the complexity of first-order logic is carried over as well, making descriptions less accessible. Also, model checking is typically performed against specifications in propositional logic rather than first-order logic. The first-order model checking problem in general has been shown to be intractable; very little work exists on algorithms for specifications in first-order logic [32].

We are concerned with simulations of stochastic models and intend to include a readable description (for both human and machine) of expected or desired simulation output in an experiment description. Taking the requirements to express properties for stochastic models and provide an easily usable notation for properties, we end up in a dilemma. On the one hand, first-order logics based methods are powerful, but quickly require the user to write and read complex formulas. On the other hand, temporal logics are well established, but lack an awareness of noise in trajectories. In this work, we remedy this disadvantage of temporal logics by prepending a data processing step to the checking of the property.

Based on a review of existing temporal logics, we decided to express properties in $\text{MITL}_{[a,b]}$ [28]. The advantages of this language include its natural integration of quantitative characteristics of temporal and non-temporal variables. We combine it with the probabilistic operator as used in CSL to add a notion of stochasticity regarding replications. Thus, we can embed $\text{MITL}_{[a,b]}$ formulas $\phi$ in $Pr_{\bowtie p}(\phi)$, where $\bowtie \in \{<, \leq, >, \geq\}$. For instance, $Pr_{\geq 0.8}(\phi)$ holds if the probability that a random simulation run of the model at hand satisfies the property $\phi$ is at least 80%.

### *2.4. Specifying data processing method*

During the execution of simulation experiments, output data is generated, collected, and further analyzed. However, the generated raw data may not be directly usable for analysis. For example, in the output trajectories of simulation experiments on stochastic models, high frequency fluctuations may overlay longer-term trends. To make such trends

apparent also at short intervals, those fluctuations have to be filtered out, leading to a smoothed trajectory. Temporal logic formulas can then be evaluated on these smoothed trajectories without being disrupted.

As a proof of concept, our approach currently only supports one data processing method for smoothing, i.e., the LOESS method (may be understood as standing for "LOcal regrESSion", as a generalization of LOWESS (Locally Weighted Scatterplot Smoothing) [33]) originally proposed by Cleveland in [34]. We integrated the implementation of the LOESS method from the Apache Commons Math library [35], with the bandwidth and the number of robustness iterations configured by the user in SESSL, and the accuracy set to $10^{-12}$. However, our approach is not limited to this method. Since we specify the data processing method using SESSL, any data processing method can be incorporated without much effort.

## 3. Proposed approach

The main purpose of our approach is to support the model development process. Each model is annotated with specifications of simulation experiments that have been executed with it. The user can take an existing model along with its experiment specifications, and extend the model.

To reuse the experiment specifications of the original model for conducting experiments on the extended model, certain adaptations may be required. Adaptation information can be derived during the model extension and specified by the user. In addition, for the expected behavioral properties of the original model, users can specify the expectation of the result, i.e., which properties should hold and which should not when checked on the extended model.

With the extended model, adaptation information and result expectation, the experiment specifications of the original model are adapted to be used with the extended model. Based on the adapted experiment specifications, simulation experiments are generated and executed for the extended model, to check whether the corresponding properties still hold or not and whether the test results are as expected. We designed and implemented TAECS (a Tool for Adaptation, Execution and Checking of Simulation experiments) to automatically realize this process.

Furthermore, the user can design and perform additional experiments on the extended model, and the specifications of those experiments are executed and checked as well. If the checking of an experiment specification fails, it is returned to the user for revision and then checked again. Each experiment specification that has been checked successfully is added as an annotation to the extended model. Figure 2 depicts the overview of the general approach.

As discussed in Section 2, SESSL already provides suitable means for model and simulator configuration, $\text{MITL}_{[a,b]}$ and CSL shall be used for property description, and a LOESS smoother for data preprocessing. Therefore, we integrated those into SESSL so that experiment specifications in SESSL can be reused as required. To check the properties, prototypical methods for statistical model checking need to be realized. Furthermore, a mechanism is required to automatically generate simulation experiments by reusing the original ones, to execute these adapted experiments, and to perform statistical model checking on the experiment results, which was realized in TAECS. We illustrate the three aspects in the following.

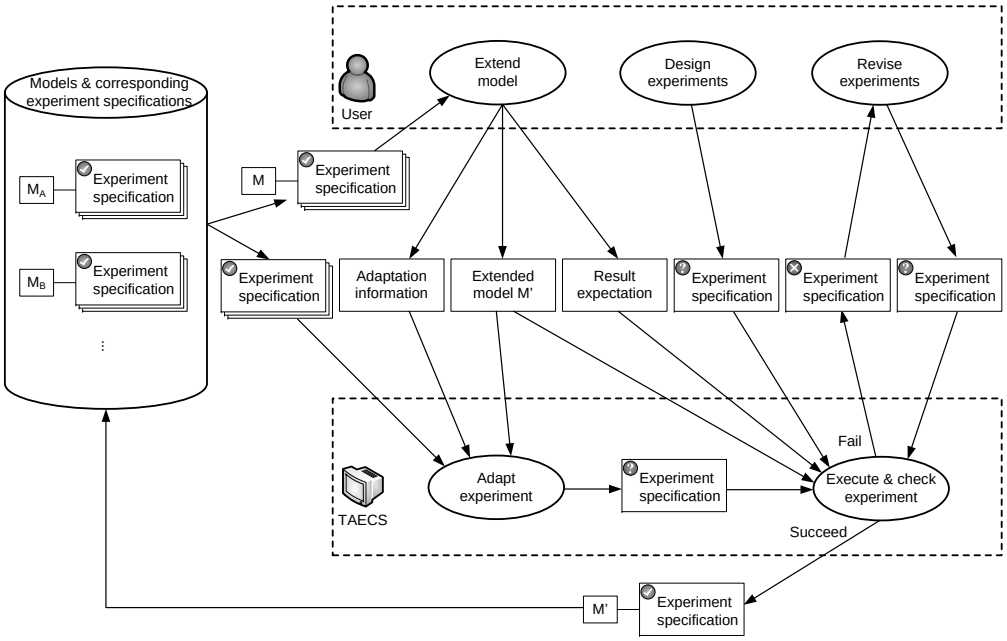



Figure 2: The approach overview. When the user extends an existing model, the corresponding experiment specifications of the original model, adaptation information, result expectation and the extended model are passed to TAECS, which first adapts the experiments and then executes and checks them. Besides, TAECS can directly execute and check experiment specifications, which are created by the user either through performing new experiments or through revising experiments that fail in previous checking. When checked successfully, the model is annotated with the experiment specification.

### *3.1. Integration into SESSL*

To enable describing the four aspects of a simulation experiment, i.e., model configuration, simulation configuration, data processing method and model behavioral property, we integrated the last two aspects into SESSL by extending the `Hypothesis` trait. An example is depicted in Figure 3, based on the experiment shown in Figure 1.

Lines 5-6 depict the model configuration, i.e., the model used for experimentation and the values of three model parameters (`nPredator`, `nPrey` and `nFood`). In the experiment specification, the configuration of model parameters is completely determined and specified by the user. In addition, a full configuration of model parameters is required in the model file, and default values are provided for each model parameter. If explicit assignments of model parameters are configured in the experiment specification, the experiment is executed with those assignments, which could be different from the default values in the model file; otherwise, the default values in the model file are used for the experiment. The simulation configuration is shown in lines 8-11. Line 13 specifies the data processing method, i.e., the LOESS method with the bandwidth parameter being 0.1 and the number of robustness iterations parameter being 0. Lines 15-17 are the property specification starting with the keyword `assume`. A probabilistic property is defined that for a simulation run the probability that the inner property is satisfied is no less than 80% (line 15). The

```
import sessl._
import sessl.james._
val exp = new Experiment extends Observation with Hypothesis {
  //model configuration
  model = "file-mlrj:/./LotkaVolterra.mlrj"
  set("nPredator" <~ 50, "nPrey" <~ 500, "nFood" <~ 100)
  //simulation configuration
  simulator = MLRulesReference()
  stopCondition = AfterWallClockTime(seconds=1) or AfterSimTime(500)
  observe("nPredator")
  observeAt(range(0, 1, 500))
  //data processing method
  dataProcessor = LoessInterpolation(0.1, 0)
  //property
  assume{(Probability >= 0.8)(
   G(0, 300, variable("nPredator") > 0) and F(300, 400, variable("nPredator") == 0)
   )}
  }
```

Figure 3: A SESSL experiment specification with four aspects: model configuration, simulation configuration, data processing method and property.

inner property, an $\text{MITL}_{[a,b]}$ expression, is specified with the operators `G` and `F`, indicating that from time 0 to time 300, the number of predator is always larger than 0 and within the time interval 300 to 400, the number of predator will eventually drop to 0.

### *3.2. The algorithm for property checking*

Statistical, simulation-based model checking techniques are frequently applied to determine whether a stochastic model satisfies a specification in the form $Pr_{\geq p}(\phi)$ [3]. More precisely, such techniques can decide whether the probability $p'$ that a random simulation run of executing the model satisfies a temporal logic formula $\phi$ is not smaller than a threshold $p \in [0, 1]$. The central idea is to employ *hypothesis testing* to make this decision based on a number of observed simulation runs [36].

We express the question whether $p' = Pr(\phi)$ is at least $p$ in two competing hypotheses. The null hypothesis $H_0 : p' < p$ states that $Pr_{\geq p}(\phi)$ does not hold; the alternative hypothesis $H_1 : p' \geq p$ states that $Pr_{\geq p}(\phi)$ holds. We can now test these hypotheses against each other. By rejecting $H_0$, we gain some evidence that $H_1$ and thus $Pr_{\geq p}(\phi)$ holds; if we fail to reject $H_0$, we gain some evidence against $Pr_{\geq p}(\phi)$. Hypothesis testing can not guarantee that our result is correct. We can, however, limit the probabilities of making a type I error (rejecting $H_0$ although it is true) and making a type II error (failing to reject $H_0$ although it is false), termed $\alpha$ and $\beta$, respectively.

Smaller values for $\alpha$ and $\beta$ correspond to a less uncertain test result. However, a low probability for one error type can only be achieved by allowing a high probability for the other error type. To make small values for $\alpha$ and $\beta$ possible, a common approach is to introduce an indifference region of width $\delta$ around $p$. Thus, the hypotheses $H_0 : p' < p - \delta$ and $H_1 : p' \geq p + \delta$ are used instead.

Similar to Sen et al. [24], we reject $H_0$ if, after executing $n$ simulation runs, more than $p \times n$ runs satisfy $\phi$. We determine a minimal $n$ based on given $p$, $\delta$, $\alpha$, and $\beta$ by increasing $n$ until the type I error probability $a \leq \alpha$ and the type II error probability $b \leq \beta$, as depicted in Algorithm 1. The error probabilities can be computed by assuming real probabilities $p' = p \pm \delta$ for given $p$, $n$, and $\delta$ [24].

**Algorithm 1** Algorithm for determining the number of replications needed to check a property. The number $X$ of simulation runs satisfying the temporal logic formula is binomially distributed, with $F$ being the cumulative distribution function.

```
calculateReplicationNumber(p, δ, α, β)
  n ← 1
  sufficient ← False
  while(not sufficient)
    n ← n + 1
    a ← P(X > p × n | X ∼ Bin(n, p − δ)) = 1 − F(n × p; n, p − δ)
    b ← P(X ≤ p × n | X ∼ Bin(n, p + δ)) = F(n × p; n, p + δ)
    sufficient ← a ≤ α ∧ b ≤ β
  return n
```

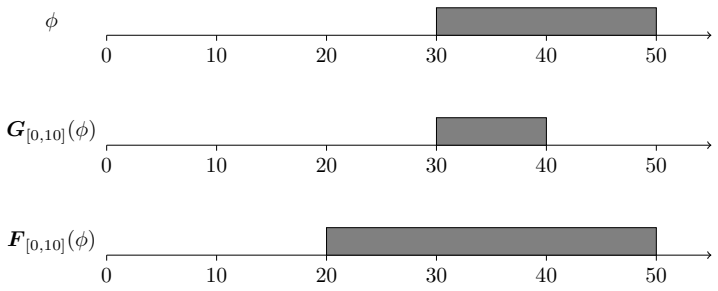



Figure 4: Example for obtaining valid intervals for formulas with temporal operators. The valid intervals for the formulas $\boldsymbol{G}_{[0,10]}(\phi)$ ($\phi$ holds during the next ten time units) and $\boldsymbol{F}_{[0,10]}(\phi)$ ($\phi$ holds somewhere in the next ten time units) shall be determined. The formula $\phi$ holds in the interval $[30, 50]$. Then the formula $\boldsymbol{G}_{[0,10]}(\phi)$ holds in $[30, 40]$, and $\boldsymbol{F}_{[0,10]}(\phi)$ holds in $[20, 50]$. Picture after Maler and Nickovic [28].

Aside from the statistical evaluation, the main work consists of determining whether a single simulation run satisfies the MITL$_{[a,b]}$ formula. We adopted the algorithm for checking MITL$_{[a,b]}$ formulas as proposed in [28]. To obtain the valid intervals for a given formula, the valid intervals for its subformulas are recursively determined (an example is shown in Figure 4). The logical operators negation, conjunction and disjunction are mapped to inverting, intersecting and joining the sets of valid intervals. The valid intervals for atomic propositions are determined directly from the trajectory data. This bottom-up approach provides an elegant and efficient algorithm for obtaining the intervals where a given complex formula holds. If one of these intervals contains the time point 0, i.e., the beginning of the trajectory, the formula holds for the observed simulation run.

Special attention has to be given to the semantics of the intervals. As defined by Maler and Nickovic, we interpret the intervals as left-closed right-open, such that they contain their start point, but not their endpoint [28]. Although this would mathematically be notated as $[t_i, t_{i+1})$, we follow the original authors and denote it as $[t_i, t_{i+1}]$. We use the set of observed time points in the trajectory as the set of possible interval borders. We define that a property holds in an atomic interval $[t_i, t_{i+1}]$ if it holds at the observed time point $t_i$. Thus, we assume that the model state does not change (regarding the property to check) between observations. Consequently, the observed trajectory should contain enough data points to justify this assumption. As "enough" is highly model-dependent,

it is crucial to interweave the configuration of model observation and MITL$_{[a,b]}$ property checking in the experiment specification, e.g., provided by SESSL.

### *3.3. Experiment adaptation, execution and checking*

In existing approaches (e.g., SED-ML and NEDL), generating and executing experiments from description scripts typically refer to one specific model, i.e., the experiments are generated and executed for the same model as the one documented in the experiment description [12, 14].

However, when the experiment specifications of the original model are reused and applied to the extended model, experiments are generated for a different model (i.e., the extended model), rather than the model described in experiment specifications (i.e., the original model). Seeing that the experiment generation of the extended model is based on the experiment specifications of the original model, some adaptations are required during experiment generation. First of all, the model to be experimented with has to be adapted and configured as the extended model, instead of the original one. Model parameters may also need to be adapted. For example, when new parameters are added, their configuration is required for experimentation. However, they are not included in the experiment specification of the original model. Also, the modeler may want to change the configuration of existing parameters, e.g., renaming or reassigning values. In addition, the model behavioral properties in the experiment specification may need adaptation as well, if some model variables are renamed during model extension. Similarly, the variables specified in the observation conditions may also need renaming.

For instance, a two-species (one prey and one predator) Lotka-Volterra model named as `LotkaVolterra` is annotated with its experiment presented in Figure 3. The user extends this model to a three-species food chain model `ThreeSpeciesLotkaVolterra` by introducing a new species, which feeds on the predator in the original model. Therefore, the extended model has one prey, one middle predator that is the predator in the original model, and one top predator that is the newly introduced species. During the model extension, in order to distinguish between two predators, the user renames the variables that are related to the predator in the original model, e.g., changing `nPredator` to `nMiddlePredator`, and also adjusts the assignment of some parameters, e.g., increasing `nFood` from 100 to 200. Meanwhile, the initial number of the top predator needs to be specified, e.g., `nTopPredator` is 30. Those changes are provided by the user as adaptation information. Consequently, in order to reuse the experiment of the original model (i.e., the one presented in Figure 3) for experimentation on the extended model, adaptation is required. The adapted experiment is shown in Figure 5. Based on the adaptation information, the model location is updated to the extended model `file-mlrj:/./ThreeSpeciesLotkaVolterra.mlrj` (line 5). The model parameter `nPredator` needs to be renamed as `nMiddlePredator`, which leads to an update in model configuration (line 6), observation condition (line 10) and property (lines 16-17). The assignment of parameter `nFood` is updated to 200 and the configuration of new parameter `nTopPredator` is added (line 6).

Currently, in our approach all the information needed for adapting the experiment is specified by the user. However, we are in the process of developing a tool to guide the user through the process of intertwining model generation, extension, and validation. Future work will aim at automatically monitoring and analyzing changes while extending a model to automatically generate suitable adaptation information. In this context, work

```
1  import sessl._
2  import sessl.james._
3  val exp = new Experiment extends Observation with Hypothesis {
4    //model configuration
5    model = "file-mlrj:/./ThreeSpeciesLotkaVolterra.mlrj"
6    set("nMiddlePredator" <~ 50,"nPrey" <~ 500,"nFood" <~ 200,"nTopPredator" <~ 30)
7    //simulation configuration
8    simulator = MLRulesReference()
9    stopCondition = AfterWallClockTime(seconds=1) or AfterSimTime(500)
10   observe("nMiddlePredator")
11   observeAt(range(0, 1, 500))
12   //data processing method
13   dataProcessor = LoessInterpolation(0.1, 0)
14   //property
15   assume{(Probability >= 0.8)(
16    G(0, 300, variable("nMiddlePredator") > 0)
17     and F(300, 400, variable("nMiddlePredator") == 0)
18    )}
19   }
```

Figure 5: The adapted SESSL experiment based on the experiment described in Figure 3.

on model transformations (e.g., [37]) as well as model version control (e.g., [38]) will be of relevance.

Furthermore, the user can state result expectations for the behavioral properties of the original model, i.e., which properties are expected to still hold and which fail after the extension, e.g., in the example of extending the Lotka-Volterra model described above, the user may specify that the property of the original model in Figure 3 would still hold for the extended model.

Therefore, in this case, which corresponds to the case A depicted in Figure 6, the extended model, SESSL experiment specifications of the original model, adaptation information and result expectation are handed over as input to TAECS. New SESSL experiment specifications are automatically generated in TAECS through adapting the original ones. Experiments are executed with the extended model based on those new experiment specifications in SESSL, and further model behavioral properties are checked against the experiments output, automatically.

The involved information can be structured as follows. We assume model $m'$ is an extension of model $m$, and $E$ is a set of experiments that have been executed with model $m$. Each experiment $e \in E$ contains a model configuration including model location $e_m$ and parameter assignment $e_a$, a simulation configuration including observation conditions $e_o$, a data processing method $e_d$, a model behavioral property $e_p$ and the replication condition $e_{rep}$. Taking the experiment presented in Figure 3 as example, $e_m$ is the model `file-mlrj:/./LotkaVolterra.mlrj` (line 5) and $e_a$ is the configuration of three parameters `nPredator`, `nPrey` and `nFood` (line 6-8). Lines 12-13 constitute observation condition $e_o$, which includes the variable to be observed (i.e., `nPredator`) and how to observe (i.e., at each time point from time 0 to 500). The LOESS method denoted as `LoessInterpolation` is the data processing method $e_d$ (line 15) and behavioral property $e_p$ is depicted in lines 17-19. In this experiment, the replication number $e_{rep}$ is not specified explicitly, but determined by the approach presented in Algorithm 1, depending on the probability defined in $e_p$ (i.e., 0.8 denoted in line 17), while in the experiment defined in Figure 1, the $e_{rep}$ indicates 10 replications (line 12).

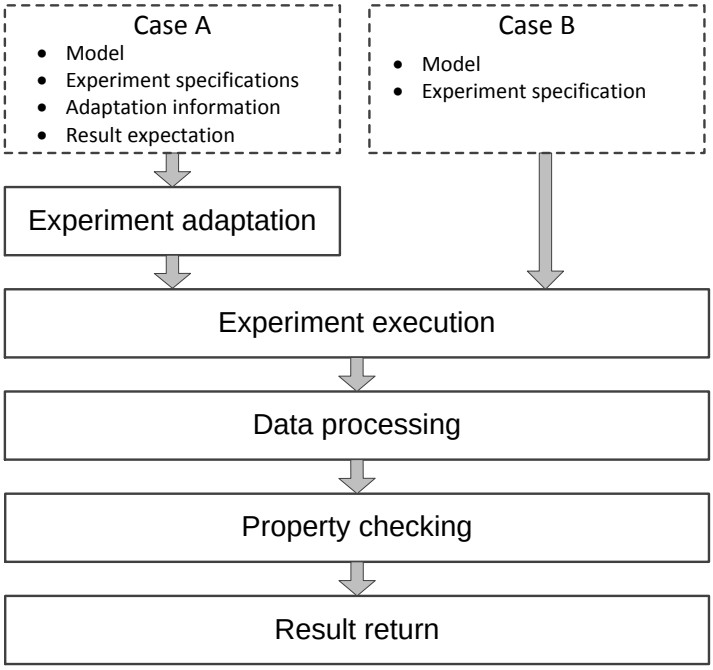



Figure 6: The working process of TAECS with two cases. In both case A and B, TAECS executes experiments based on experiment specifications. And then corresponding data processing is performed on the raw experiment results. Afterwards, the processed data are checked against specified properties. In case A, TAECS takes experiment specifications, the model, adaptation information and result expectation as input, and the experiment specifications are first adapted, while in case B it only takes the experiment specifications and the model as input, and the experiment specifications are directly executed and checked without adaptation.

In the adaptation information $\kappa$, users can specify the change of parameter configuration $\kappa_a$ and the change of model variables $\kappa_{names}$. In the example of extending Lotka-Volterra model described above, $\kappa_a$ includes assigning parameter `nFood` to 200 and `nTopPredator` to 30, and $\kappa_{names}$ is a mapping from variable name `nPredator` to `nMiddlePredator`. Additionally, users can specify the expected result of checking the property on the extended model $m'$ in the result expectation $r$, i.e., for each experiment $e$ in $E$, expecting that property $e_p$ will hold or not.

The procedure of TAECS in case A is depicted in Algorithm 2. The algorithm iterates all experiments of the original model $m$. For each experiment $e \in E$, first of all, the experiment is adapted based on the adaptation information $\kappa$ (lines 9-14). The model location $e_m$ is updated (line 10). The configuration of model parameters $e_a$, the variable names in the property $e_p$ and the variable names in observation conditions $e_o$ are updated as well (line 12). Apart from necessary adaptation, all the other aspects of the experiment, e.g., the simulator, simulation stop time and the data processing method $e_d$, remain the same. Then the new experiment $e'$ is generated (line 14).

In stochastic simulation, replication numbers have an important influence on simulation results, and therefore have an impact on the evaluation of properties. To gain confidence

**Algorithm 2** Algorithm for reusing experiment specifications.
$m'$: the extended model.
$E$: experiments executed for model $m$.
$\kappa$: adaptation information.
$r$: result expectation.
$\delta, \alpha, \beta$: parameters required for calculating replication number.

```
1  // Test result
2  res ← true
3  // Return set for successful experiments
4  E+ ← ∅
5  // Return set for failed experiments
6  E− ← ∅
7  for each experiment e ∈ E
8  /*** Experiment adaptation */
9  // Update model location to the extended model
10   e_m' ← updateModel(e_m, m')
11 // Update parameter configuration, property and observation variables
12   e_a', e_o', e_p' ← updateExperiment(e, κ)
13 // Generate experiment for the extended model
14   e' ← generateExperiment(e_m', e_a', e_o', e_p', e)
15 /*** Experiment execution */
16 // Determine replication number (see Algorithm 1)
17   e'_rep ← calculateReplicationNumber(e'_p, δ, α, β)
18 // Execute experiment
19   Y ← run(e')
20 /*** Data Processing */
21   Y' ← processData(Y, e'_d)
22 /*** Property checking */
23   result ← check(e'_p, Y')
24 /*** Result return */
25 // Add successful experiment
26   if(result is true)
27     E+ ← E+ ∪ {e'}
28 // Add failed experiment
29   else
30    E− ← E− ∪ {e'}
31 // Compare and aggregate adherence to result expectation
32   res ← res ∧ (result equal getExpectation(r, e))
33 end for
34 return res, E+, E−
```

in checking the probabilistic properties of the extended model, a specific strategy to determine the replication number of experiments is required. Based on the specified property, we use the approach presented in Algorithm 1 to determine the replication number, which depends on the probability specified in the property (see discussion in Section 3.2), by calling the function `calculateReplicationNumber` implemented in Algorithm 1 (line 17). In our tool TAECS, $\delta, \alpha, \beta$ are all configured with default value 0.05; however, they can be specified by the user as well.

Afterwards, the new experiment is executed, resulting in output trajectories $Y$ (line 19). New trajectories $Y'$ are obtained by applying the data processing method $e'_d$ on the trajectories $Y$ (line 21). The new trajectories $Y'$ are checked against the property $e'_p$(line 23) and the checking result $result$ is compared with the result expectation $result'$ specified by the user.

When all experiment specifications are adapted, executed and checked, two types of information are returned to the user. On the one hand, the result expectation specified by the user is tested against the checking result of each experiment specification, and the test result is returned to the user, which corresponds to the $res$. When the checking results of all experiments are as expected, the test result is true; if the checking result of one experiment does not match the expectation, false will be returned (as depicted in line 32). As the result expectation is derived based on model extension by the user, with this type of information, the user can find out whether the extended model actually behaves as expected, and if not, the extension performed on the original model may need to be revised. On the other hand, regardless of result expectations, the checking result of each experiment specification is recorded and returned, which corresponds to $E+$ and $E-$ (lines 25-30). Those experiment specifications that are checked successfully are returned to be annotated with this model; those whose checking result is false are returned to the user for revision.

Besides reusing experiment specifications, the user can design and perform additional experiments on the extended model, or revise the experiments that fail in the checking. In both situations, an experiment specification and related model are passed to TAECS, without adaptation on the experiment specification. This corresponds to the case B as shown in Figure 6. Taking only the experiment specification and the model as input, the algorithm for the procedure in case B is similar to Algorithm 2, without the experiment adaptation and comparison to result expectation.

### *3.4. Implementation*

The structure of the implementation of our approach is shown in Figure 7. It consists of three layers.

The first layer serves as the user interface, where the user can specify the model location and simulation experiments, and define the necessary information for adaptation and the result expectation when experiment specifications need to be adapted. All of this information is passed as input to the second layer. Currently, we employ the extended SESSL as user interface. However, we are also in the process of developing a graphical user interface, which allows the user to load the model and experiment specifications, as well as specifying adaptations and result expectations more conveniently.

Based on the information handed over by the first layer, the second layer conducts experimentation on the extended model with or without adaptation, and performs

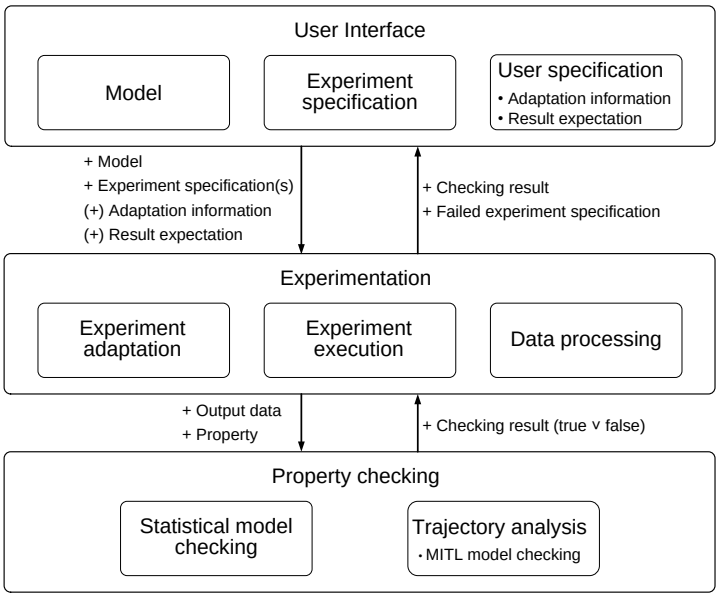



Figure 7: The architecture of our prototype implementation. As the user interface, a SESSL extension is used in the upper layer where the user can specify the model, experiment specifications, adaptation information and result expectation. The middle layer conducts experiments and performs data processing, with two types of input, i.e., the model, experiment specifications, adaptation information and result expectation, or only the model with the experiment specification. With output data and property, the lower layer checks the property and returns checking results. The middle layer and the lower layer constitute TAECS.

corresponding data processing on the raw simulation output. After data processing, the third layer is triggered by the second layer to perform property checking, i.e., the simulation output is checked against the formalized properties. So far, we have implemented one checking algorithm for $\text{MITL}_{[a,b]}$ and one statistical model checking method, as described in Section 3.2. Additional model checking approaches can be included as well.

TAECS consists of the second layer and the third layer, and is integrated into SESSL as well. As SESSL supports different simulation systems by creating bindings, exchanging modeling formalisms and simulation methods are implicitly supported by our approach.

## 4. Case Study

To illustrate our approach, we successively developed models that describe the process of receptor ligand binding and subsequent signaling events (as shown in Figure 8) at various levels of detail. Three models were developed one after another by extension, following the previously described approach. At first a basic model (M0) describing the formation of a simple ternary-complex at the membrane was developed and simulation experiments were performed.

These experiments along with the corresponding behavioral properties of the model (i.e., the ternary-complex model) were described with SESSL. Next, an extended model (ME1) was developed by adding the process of endocytosis to the ternary-complex model

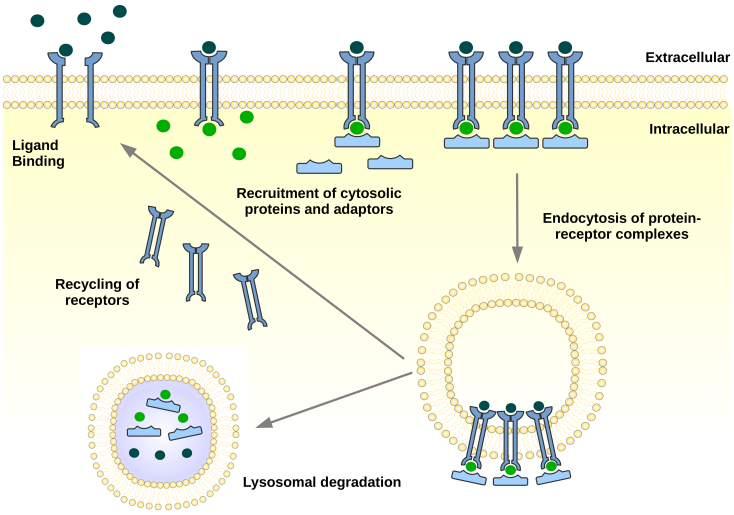



Figure 8: The ligand receptor pathway.

(M0). Based on the experiment specifications of M0, we generated experiments for the endocytosis model (ME1) and checked the model behavioral properties using our approach. Repeating the same process, we further extended the endocytosis model (ME1) by means of an recycling model (ME2).

### *4.1. Background - Signaling events induced by receptor-ligand binding*

The binding of a ligand to a membrane-integral receptor plays a pivotal role in cell communication. It is the initial step in cellular signal transduction and provides the opportunity to transmit signals and information from the outside of the cell into the inside. Once a ligand-receptor complex is successfully established, an intracellular signaling cascade is initiated, which eventually leads to the input-specific response of the cell, such as the specific expression of a target gene. The strength/effectiveness of the signal response depends, among other, on how many ligands can be bound and, in particular, how stable the complex is, i.e., how long the binding between receptor and ligand lasts.

However, the stability as well as the function of the receptor complex is greatly affected by other membrane-associated (peripheral) proteins that interact with the ligand-receptor complex. Thereby the interaction may eventually lead to the formation of a ternary complex, consisting of ligand, receptor and peripheral protein. Common examples of proteins involved in the formation of ternary complexes are G-proteins, coated pit adaptors, cytoskeletal elements or other receptors [39]. The impact of such receptor coupling interactions and the formation of a ternary complex is two-fold. On the one hand, the association and dissociation rates for receptor-ligand binding can vary significantly between binary and ternary complexes —a fact that has to be taken into close consideration when analyzing experimental data. Renown examples for this effect are receptor/G-protein coupling and EGF receptor/adaptor coupling.

On the other hand, the interaction with membrane-associated proteins induces further signaling events, like internalization (endocytosis) and recycling processes or receptor

accumulation. With our exemplary model, we aim to capture the essential receptor-ligand binding kinetics depending on the previously described receptor mediated events, i.e., receptor-protein coupling, internalization and recycling.

*4.2. Modeling approach and simulation method*

The models of our case study are described in ML-Rules. ML-Rules is a rule-based language developed for supporting the modeling of cell biological systems [17, 40, 18]. It supports nested rule schemata, the hierarchical dynamic nesting of species (e.g., to express processes like endocytosis), the assignment of attributes and solutions to species at each level, and a flexible definition of reaction rate kinetics. As ML-Rules allows the compact description of rather complex models, means for a more efficient execution were developed, e.g., approximate and adaptive algorithms [41] or strategies to automatically select and configure suitable simulators [42]. Within our experiments the simulation algorithm described in [17, 18] has been used, named as `MLRulesReference`. The modeling and simulation framework JAMES II was exploited as the simulation system which actually executes the experiments specified in SESSL.

Throughout our case study we used the method presented in [24] as discussed in Section 3.2 to determine replication numbers with $\alpha = \beta = \delta = 0.05$.

*4.3. Basic model (M0) - Ternary-complex formation*

We started our case study with the development of the ternary-complex model, as depicted in Figure 9, and would successively extend it with endocytosis and recycling processes. Accordingly the most basic version of the model contains four different species: ligand (L), receptor (R), adaptor protein (X) and a representation for protein complexes (C), which can have different states, depending on how many compounds are bound. For simplicity we disregard the scenario where cytosolic proteins bind the receptor first and after that the ligand is bound. We motivate this assumption by the fact that the reaction rate constants for this scenario are significantly lower, therefore can be neglected in the model.

The ternary-complex model is of general nature, but can be parametrized and fitted to match the dynamics of specific pathways. In this case study we aim to reproduce EGF / EGFR binding kinetics that were experimentally determined in fibroblasts. In the wet-lab experiments, cells were stimulated for a certain time with EGF and the binding kinetics were observed by measuring the amount of ligand/receptor complexes at different time points. After a given time, cells were washed and EGF was completely removed from the system. For this system the reaction rate constants of all reactions are available

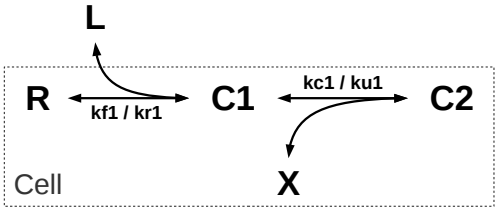



Figure 9: The ternary-complex model (M0).

and were used to parametrize our ternary model (see supplementary file). In our model we applied a concentration of $5 \times 10^{-10}$ as EGF stimulus for a time period of 15 minutes, i.e., after 15 minutes all remaining EGF molecules are removed, similar to the wet-lab experimental set-up [43].

We chose the number of receptor (R), adaptor protein (X) and the number of a representation for protein complexes (C) as species of interest to explore behavior properties of the ternary-complex model (M0). To run simulation experiments, we configured the model parameters as follows:

```
nR: 6e04,   nL: 1.05e09,   nX: 2.4e04,
```

where `nR, nL` and `nX` denote the initial number of species R, L and X, respectively.

To investigate the dynamics of the three species of interest, we set the simulation stop time as 2000 simulation time units (with one time unit corresponding to one second in the wet-lab experiments). The model was formalized in ML-Rules [17] and we chose the simulation algorithm presented in [18], i.e., `MLRulesReference`, for simulation experiments. Since ML-Rules is based on Continuous-Time Markov Chain semantics, the results are stochastic and therefore multiple replications are required. To explore the behavior of the model, we firstly configured each simulation experiment with replication number being 200. Simulation experiments were executed and the number of the three species (R, C and X) were observed, named as variable `Cell/R`, `Cell/C` and `Cell/X` in the model, respectively. The simulation trajectories of the observed variables are shown in Figure 10 (a), (b) and (c), and are in good agreement with wet-lab experimental results [43]. Due to the stochasticity, some "noise" exists in the generated simulation trajectories. We applied the LOESS method to the original trajectories, with the bandwidth parameter being 0.1 and the number of robustness iterations parameter being 0, and the smoothed trajectories are shown in Figure 10 (d), (e) and (f). However, the LOESS method may have an impact on the initial value in the trajectories, such as decreasing or increasing the values, e.g., in Figure 10 (d) and (a), or (e) and (b). However, this decreasing or increasing of values does not affect the model behavioral properties we inspected in our case study. This also applies to the experiments of the two extended models ME1 and ME2, which will be discussed in Section 4.4 and 4.5, i.e., Figure 13 and 17.

Through analyzing the simulation results, we derived the model behavioral properties regarding the three species as follows:

- Species `Cell/R`: with a probability of no less than 0.75, the species number remains within the range [57000, 58000] over the time interval [300, 800], and reaches a steady state over the time interval [1200, 2000] with the amount being in the range [59750, 59850].

- Species `Cell/C`: with a probability of no less than 0.95, the species number remains within the range [2000, 2300] over the time interval [300, 800], and reaches a steady state over the time interval [1300, 2000] with the amount being in the range [150, 250].

- Species `Cell/X`: with a probability of no less than 0.85, the species number decreases over the time interval [0, 800] and increases over the time interval [1300, 2000].

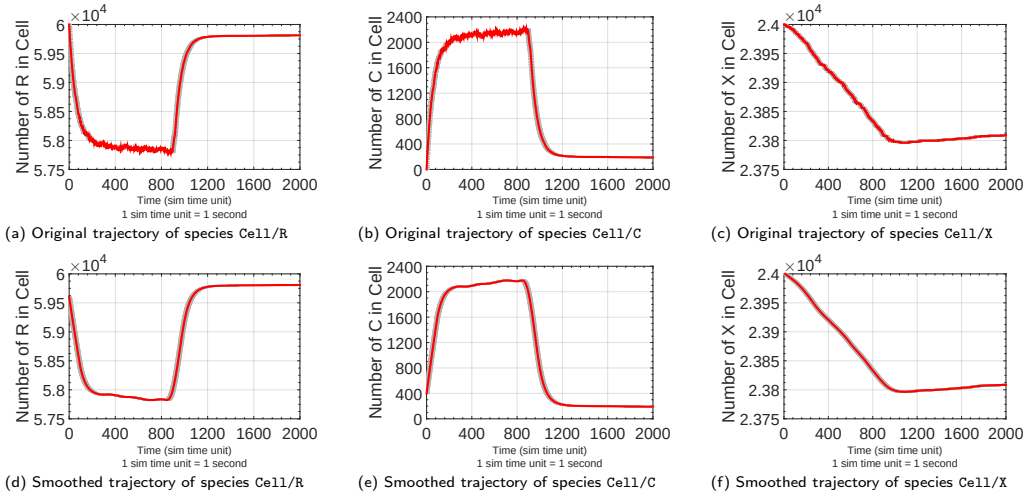



Figure 10: Simulation results of the ternary-complex model (M0). The mean trajectories (in red) with the standard error (in gray error bars) are shown. For all trajectories, the simulation stop time is after 2,000 time units. (a), (b) and (c) are the original trajectories from simulation experiments; (d), (e) and (f) are the smoothed trajectories using the LOESS method on the original ones, respectively. Please note, as a result of the smoothing the initial values of trajectories (d) and (e) slightly deviate from the original trajectories (a) and (b). As explained in the text, this deviation does not affect the considered model behavioral properties, hence does not bias our test results.

Based on the simulation set-up and derived properties, we specified the experiments on model M0 in SESSL for each species of interest. For all species, the model configuration, simulation configuration and data processing method are the same, while the specifications of the model behavioral property are different. We present the three experiment specifications in Figure 11, with complete SESSL experiment specification regarding species `Cell/R` shown in Figure 11 (a), including the model configuration, simulation configuration, data processing method and model behavioral property. For simplicity, we only depict the property specifications regarding species `Cell/C` and `Cell/X`, as shown in Figure 11 (b) and (c). To describe properties that a variable increases or decreases, we use the first order deviation of the variable, which is specified as `d("variableName")` in SESSL. For example, a property that the number of species `Cell/X` decreases is expressed as `d("Cell/X") <= 0` in SESSL, which means the first order deviation of the number of `Cell/X` is less than 0, as shown in Figure 11 (c).

We tested each of the three experiment specifications on the ternary-complex model (M0) using TAECS, which corresponds to case B in Figure 6. The necessary replication numbers for checking the properties of `Cell/R`, `Cell/C` and `Cell/X` were automatically determined, i.e., 205, 77 and 154, respectively. All experiment specifications were checked successfully, and therefore the ternary-complex model M0 was annotated with the three experiment specifications.

```
import sessl._
import sessl.james._
val exp = new Experiment with Observation with Hypothesis {
  model = "file-mlrj:/./TernaryComplexModel.mlrj"
  set("nR" <∼ 6e04, "nL" <∼ 1.05e09, "nX" <∼ 2.4e04)
  simulator = MLRulesReference()
  stopTime = 2000
  observe("Cell/R")
  observeAt(range(0, 1, 2000))
  dataPreProcessor = LoessInterpolation(0.1, 0)
  assume{(Probability >= 0.75)(
   G(300, 800, (variable("Cell/R") >= 57000) and (variable("Cell/R") <= 58000))
    and G(1200, 2000, ((variable("Cell/R") >= 59750) and variable("Cell/R") <= 59850))
    )}
  }
```

(a) Experiment specification regarding species `Cell/R`

```
assume{(Probability >= 0.95)(
 G(300, 800, variable("Cell/C") >= 2000 and variable("Cell/C") <= 2300)
  and G(1300, 2000, variable("Cell/C") >= 150 and variable("Cell/C") <= 250)
 )}
```

(b) Specification of model behavioral property regarding species `Cell/C`

```
assume{(Probability >= 0.85)(
 G(0, 800, d("Cell/X") <= 0) and G(1300, 2000,d("Cell/X")>=0)
 )}
```

(c) Specification of model behavioral property regarding species `Cell/X`

Figure 11: Experiment specifications of the ternary-complex model (M0) in SESSL.

### *4.4. Model extension 1 (ME1) - Endocytosis*

The binding of adaptor proteins, such as clathrin, to the ligand-receptor complex often induces the endocytosis of the complex. This means the receptor complex is internalized into the cytosol as part of an endosome. As a result, ligand, receptor and adaptor proteins are not available at the cell surface anymore. This process is the subject of the first extension of our basic ternary-complex model (M0). Accordingly, we introduced a first order reaction that describes the internalization of the ternary complex into a cytoplasmic endosome with a certain rate constant $ke1$ (see Figure 12).

The extension of the model directly influences the validity of the assumptions made for the ternary-complex model. On the one hand, species that are encapsulated in the endosomes change their location (i.e. `Cell/C` → `Cell/Endosome/C`). As a result, we assume that the maximum amount of `Cell/C` will be lower in the endocytosis model (ME1) than in the ternary-complex model (M0), as a portion of the complexes C will also be located within endosomes (`Cell/Endosome/C`). We thus expected that the properties defined and checked for `Cell/C` in the ternary-complex model will not hold in the endocytosis model. On the other hand we also have to consider the species that are transferred to the endosome as part of the complex, i.e., `Cell/R` and `Cell/X`. These species are not available for the backward reactions (*kr1, ku1*), hence their amount should decrease by the number of complexes internalized. Accordingly, their property should be violated in the endocytosis model (ME1) as well.

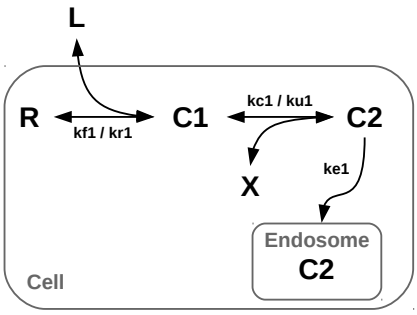



Figure 12: The endocytosis model (ME1).

*4.4.1. Reusing simulation experiments of the ternary-complex model*

To inspect the behavior of the first extended model (ME1), we reused the simulation experiments conducted on the ternary-complex model (M0). The model location needs to be adapted from the original model (M0) to the extended model (ME1). Since there is no change in model parameters, no adaptation is necessary for their configuration. We performed tests using our approach, which corresponds to case A in Figure 6, with the input of the extended model ME1, experiment specifications shown in Figure 11, adaptation information (i.e., the new model location) and result expectation (i.e., all the three experiments would fail). Experiments were generated and checked by adapting the existing ones. The testing results are described in Table 1, and the simulation trajectories are shown in Figure 13 (a), (b) and (c).

Based on our previous assumption, the properties regarding all three species in the ternary-complex model should be violated in the endocytosis model. However, the testing results revealed that only the properties regarding `Cell/C` and `Cell/X` are violated, whereas the property regarding `Cell/R` is still valid in the endocytosis model (see Table 1 and Figure 13), which is contrary to our previous assumption.

After revising our simulation experiments we found that this apparent contradiction can be resolved by applying a longer simulation run time. In fact, for simulation experiments on the ternary-complex model, at the end of the simulation the trajectory of `Cell/R` is not in steady state but increases slowly, with a sufficiently long simulation run time (30,000 instead of 2,000 time units), as shown in Figure 14 (a). However, in the trajectory from simulation experiments on the endocytosis model with 30,000 time units, the number of species `Cell/R` reaches a steady state and is lower than that on the ternary-complex model at the end of simulation, as shown in Figure 14 (d). Therefore, the behavioral property regarding species `Cell/R` exhibited in the endocytosis model does differ from

Table 1: Results of testing on the endocytosis model (ME1) by reusing experiments of the ternary-complex model (M0).

| Property | Replication number | Assumed result | Test result |
|---|---|---|---|
| **Cell/R** | 205 | false | true |
| **Cell/C** | 77 | false | false |
| **Cell/X** | 154 | false | false |

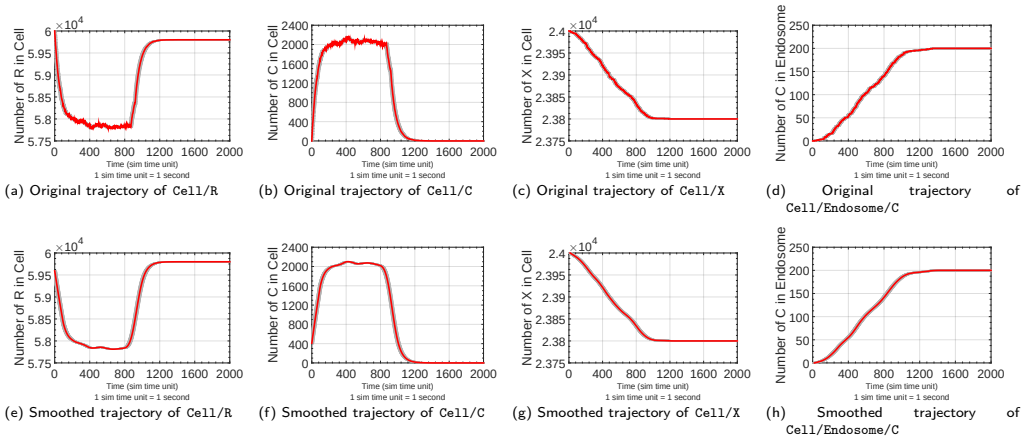



Figure 13: Simulation results of the endocytosis model (ME1). The mean trajectories (in red) with the standard error (in gray error bars) are shown. For all trajectories, the simulation stop time is after 2,000 time units. (a), (b), (c) and (d) are the original trajectories from simulation experiments; (e), (f), (g) and (h) are the smoothed trajectories using the LOESS method on the original ones, respectively.

that exhibited in the ternary-complex model, which is in agreement with our assumption.

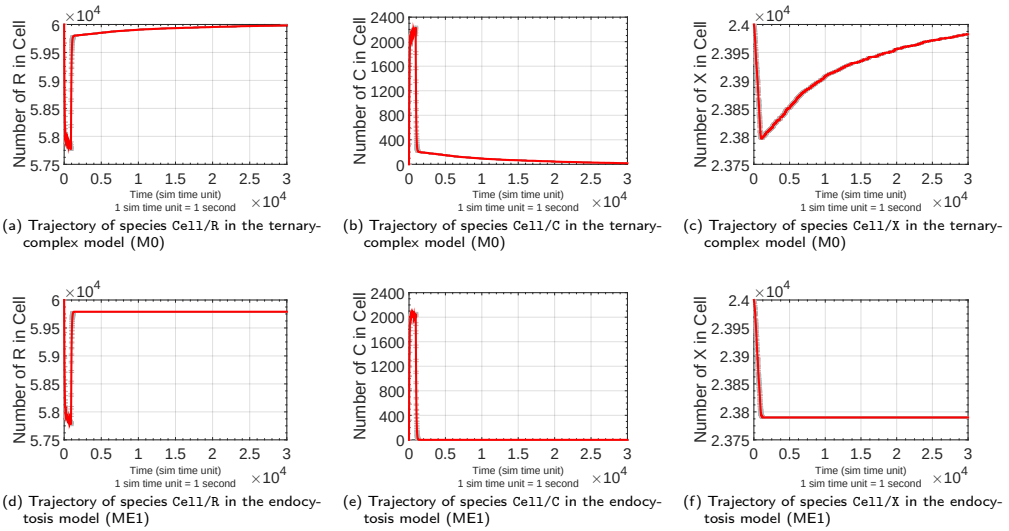
*4.4.2. Annotating experiment specifications for further extension*

While the experiment specification regarding species `Cell/R` was checked successfully and therefore annotated with model ME1, as shown in Figure 15 (a), the two other experiment specifications of the ternary-complex model (M0) failed in the checking, which need to be revised. From the simulation trajectories shown in Figure 14 (d), (e) and (f), we can see that in the endocytosis model, there is no significant change in the trajectories after 2,000 time units and the model behavior reaches steady state. As the main behavioral properties of this model can be reflected in the trajectories within 2000 time units, for better illustration in the paper, we chose to stay with 2000 time units as the simulation stop time for the subsequent simulation experiments.

Besides, according to our extension, the receptor complex in the endocytosis is of interest. Therefore we performed simulation experiments to observe the species `Cell/Endosome/C` and the simulation trajectories are shown in Figure13 (d) and (h). Based on the simulation results, we derived three new properties of the endocytosis model (ME1), described as below.

- Species `Cell/C`: with a probability of no less than 0.75, the species number reaches a steady state over the time interval [300, 800] with the amount being in the range [1900, 2200], and also reaches a steady state over the time interval [1600, 2000] with the amount being 0.

- Species `Cell/X`: with a probability of no less than 0.75, the species number decreases over the time interval [0, 900] and reaches a steady state over the time interval [1200, 2000] with the amount being in the range [23700, 23800].




(a) Trajectory of species `Cell/R` in the ternary-complex model (M0)

(b) Trajectory of species `Cell/C` in the ternary-complex model (M0)

(c) Trajectory of species `Cell/X` in the ternary-complex model (M0)

(d) Trajectory of species `Cell/R` in the endocytosis model (ME1)

(e) Trajectory of species `Cell/C` in the endocytosis model (ME1)

(f) Trajectory of species `Cell/X` in the endocytosis model (ME1)

Figure 14: Simulation trajectories of the ternary-complex model (M0) and the endocytosis model (ME1) with long simulation run time (30,000 time units). The mean trajectories (in red) with the standard error (in gray error bars) are shown. (a), (b) and (c) are the trajectories from the ternary-complex model; (d), (e) and (f) are the trajectories from the endocytosis model.

- Species `Cell/Endosome/C`: with a probability of no less than 0.75, the species number increases over the time interval [0, 900] and reaches a steady state over the time interval [1200, 2000] with the amount being in the range [200, 250].

For all of the three species, the model configuration, the simulation configuration and the data processing method remain the same as that in the experiment specification regarding species `Cell/R`. We specified the simulation experiments and corresponding properties regarding the three species in SESSL, and present the property specifications in Figure 15 (b), (c) and (d). Furthermore, we checked the experiment specifications with TAECS using the procedure of case B shown in Figure 6, and all of them succeeded. Therefore, the endocytosis model (ME1) was annotated with all four simulation experiment specifications, i.e., experiments regarding species `Cell/R`, `Cell/C`, `Cell/X` and `Cell/Endosome/C`, as depicted in Figure 15.

### *4.5. Model extension 2 (ME2) - Recycling*

Typically, during endocytosis the internalized receptor complex gets dissociated. Subsequently the receptor is recycled back to the membrane, while the ligand and adaptor protein are transferred to a lysosome, where they are degraded, as shown in Figure 16.

We further extended the endocytosis model (ME1) by this process, yielding the second extended model ME2, i.e., the recycling model. As a result, the concentration of the receptors in the cell should be slowly restored after the washing step, hence resemble the dynamics of the ternary-complex model, but not the endocytosis model.

In contrast, the ternary complex `Cell/C` and its remaining components, i.e., `Cell/X`, are still encapsulated from the cell, i.e., its dynamics do not change compared to the

```
import sessl._
import sessl.james._
val exp = new Experiment with Observation with Hypothesis {
  model = "file-mlrj:/./EndocytosisModel.mlrj"
  set("nR" <~ 6e04, "nL" <~ 1.05e09, "nX" <~ 2.4e04)
  simulator = MLRulesReference()
  stopTime = 2000
  observe("Cell/R")
  observeAt(range(0, 1, 2000))
  dataPreProcessor = LoessInterpolation(0.1, 0)
  assume{(Probability >= 0.75)(
   G(300, 800, (variable("Cell/R") >= 57000) and (variable("Cell/R") <= 58000))
    and G(1200, 2000, ((variable("Cell/R") >= 59750) and variable("Cell/R") <= 59850))
  )}
}
```

(a) Experiment specification regarding species `Cell/R`

```
assume{(Probability >= 0.75)(
 G(300, 800, variable("Cell/C") >= 1900 and variable("Cell/C") <= 2200)
  and G(1600, 2000, variable("Cell/C") == 0)
)}
```

(b) Specification of model behavioral property regarding species `Cell/C`

```
assume{(Probability >= 0.75)(
 G(0.0, 900, d("Cell/X") <= 0) and
  G(1200, 2000, variable("Cell/X") >= 23700 and variable("Cell/X") <= 23800)
)}
```

(c) Specification of model behavioral property regarding species `Cell/X`

```
assume{(Probability >= 0.75)(
 G(0, 900, d("Cell/Endosome/C") > 0) and
   G(1200, 2000, variable("Cell/Endosome/C") >= 200 and variable("Cell/Endosome/C") <= 250)
)}
```

(d) Specification of model behavioral property regarding species `Cell/Endosome/C`

Figure 15: Experiment specifications of the endocytosis model (ME1) in SESSL

endocytosis model. However, since species C in the endosome is being degraded, it cannot aggregate in the endosome as it did in the endocytosis model. Thus, when checking whether the properties defined for the endocytosis model also hold for the recycling model, we expect that the properties regarding species `Cell/R` and `Cell/Endosome/C` do not hold, while the properties regarding species X and species C in Cell still hold.

We applied our approach on the recycling model, by reusing the simulation experiment specifications of the endocytosis model depicted in Figure 15. The four experiment specifications, together with the newly extended model ME2, adaptation information (only includes the new model location, as no adaptation is needed for model parameters and properties) and result expectation were passed to TAECS. Using the procedure of the case A presented in Figure 6, experiments were automatically generated by adapting the experiment specifications of ME1 and executed on the recycling model ME2, and corresponding properties were checked.

The test results (Table 2) show the experiments regarding species `Cell/C` and `Cell/X` were checked successfully, while the experiments regarding species `Cell/R` and

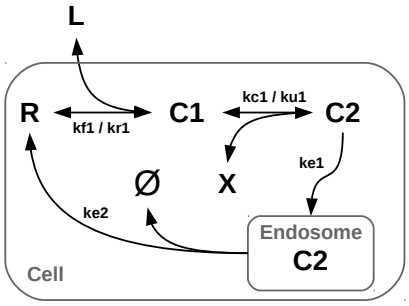
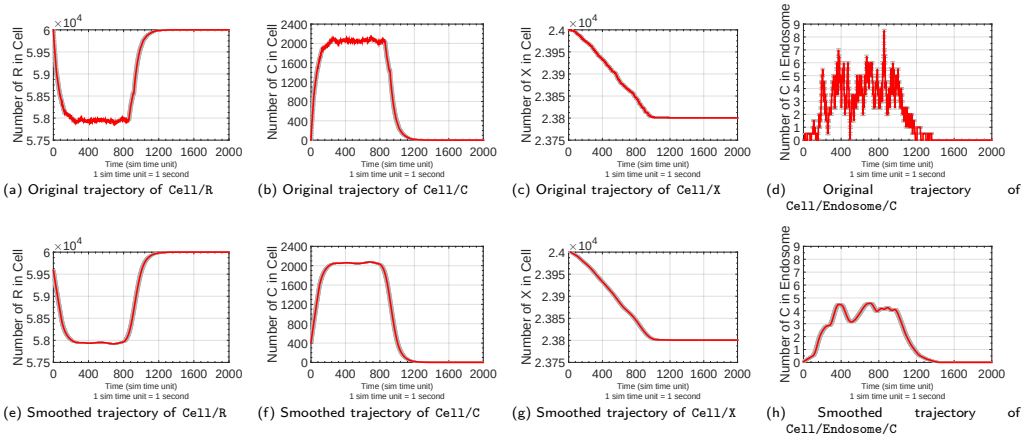



Figure 16: The recycling model (ME2).




(a) Original trajectory of `Cell/R` (b) Original trajectory of `Cell/C` (c) Original trajectory of `Cell/X` (d) Original trajectory of `Cell/Endosome/C`

(e) Smoothed trajectory of `Cell/R` (f) Smoothed trajectory of `Cell/C` (g) Smoothed trajectory of `Cell/X` (h) Smoothed trajectory of `Cell/Endosome/C`

Figure 17: Simulation results of the recycling model (ME2). The mean trajectories (in red) with the standard error (in gray error bars) are shown. For all trajectories, the simulation stop time is after 2,000 time units. (a), (b), (c) and (d) are the original trajectories from simulation experiments; (e), (f), (g) and (h) are the smoothed trajectories using the LOESS method on the original ones, respectively.

`Cell/Endsome/C` failed. Our assumptions are indeed reflected in the test results, i.e., the model extension worked as intended. The simulation trajectories are shown in Figure 17. The model ME2 can be annotated with the experiment specifications that were successfully checked, whereas the failed ones need revision.

## 5. Discussion

In recent years, more and more researchers employ model checking techniques to support model validation, such as [44, 45, 46]. The model behavioral properties of interest are formalized with temporal logics and the experiment results of models are checked against those properties.

In those approaches, only interesting properties of the model behavior are explicitly specified so that they can be checked in model validation. However, model validation

Table 2: Results of testing on the recycling model (ME2) by reusing experiments of the endocytosis model (ME1).

| Property | Replication number | Assumed result | Test result |
|---|---|---|---|
| **Cell/R** | 205 | false | false |
| **Cell/C** | 205 | true | true |
| **Cell/X** | 205 | true | true |
| **Cell/Endosome/C** | 205 | false | false |

is always related to the experiment (cf. [47] p.5). Thus, we take a step further by bringing simulation experiments and model behavioral properties together. We describe experiments explicitly and unambiguously in SESSL, use $\text{MITL}_{[a,b]}$ to specify model behavioral properties and document them together with the model. Therefore, the simulation experiments that are conducted to validate models can be reproduced and the same behavioral properties that are checked for validation can be checked again.

Other approaches exist to document models and simulation experiments. For example, [48] proposes to associate models with their simulation experiments in graph databases, which aims to provide benefits in model storage and retrieval. In [49], an on-line resource was presented, where models and simulation experiments (protocols) are encoded separately. Different simulation experiments can be applied to the model for inspection. The focus is on analyzing and comparing behavior of different models under different experimental conditions.

In our approach, we annotate models and the simulation experiments that have been executed with it, as well as the key behavioral properties the model exhibits in the experiments. Our approach aims at supporting the process of model extension. As a model is extended, the model behavioral properties, which are checked with simulation experiments on the original model, may be interesting for inspecting the behavior of the extended model. Thus, those simulation experiments need to be repeated on the extended model. By explicitly specifying simulation experiments, our approach reuses the experiment specifications of the original model to automatically perform experiments on the extended model and check the properties, and therefore provides insights into the behavior of the extended model.

Apart from performing automatic experimentation by reusing experiment specifications, our tool TAECS can assist the user in directly conducting experiments on models and documentation of experiments as well.

The presented approach bears similarities to the idea of *regression testing* in software development, i.e., upon a modification of the software under development, previously created software test cases are reused to ensure that the modification does not introduce any new faults. Two types of regression testing are distinguished: *progressive* regression testing, where the specification of the software is changed due to new requirements or new features added to the software, and *corrective* regression testing, where only design decisions and instructions of the software are involved and the specification remains unmodified [50]. Progressive regression testing, i.e., modifying software and its specification, corresponds to extending a model and updating its behavioral properties in our approach. In the regression testing process, existing test cases are often analyzed to identify the *obsolete* test cases, which are defined as those that "can no longer be used", e.g., a test case whose input/output relation becomes incorrect due to the specification

modification [50]. The obsolete test cases are not tested for the modified software. In our approach, model extension may result in the change of model behavior so that some behavioral properties that used to hold for the original model would be violated for the extended model. However, instead of eliminating those experiments, we still reuse them to generate experiments for the extended model. In addition, we allow the user to specify the expected outcome and compare it to the checking results. Many efforts have been devoted to support regression testing, e.g., on selection of test cases [51]. So far, the focus has been mostly on corrective regression testing [52].

Our approach is mainly concerned with stochastic models. When conducting simulation experiments with stochastic models, stochasticity has to be taken into account. In addition to the requirement for multiple replications, the stochastic fluctuations contained in individual simulation trajectories need to be considered as well for property checking [53]. One way to separate short-term fluctuations from interesting trends in the trajectory is the LOESS method, as described in detail in [54].

In the case study, we employed ML-Rules for the models and one of its simulation algorithm to run experiments. However, our approach is independent of simulation systems, modeling formalisms and simulation methods. Although currently only $\text{MITL}_{[a,b]}$ and CSL are used to specify model behavioral properties and one checking method is implemented, our approach is open to extension with other temporal logics or specification languages and model checking algorithms, to express more complex properties and different types of properties, such as robustness analysis (see [55], [56]).

Even though our approach was applied in the field of systems biology for the case study, it is not restricted to this and can be used to support model development in other areas as well.

## 6. Conclusions

In this paper, we present an approach to support model extension based on reusing simulation experiments. We rely on an explicit specification of simulation experiments of the original model and statistical model checking mechanisms to automatically conduct experimentation on the extended model. The simulation experiments are described in SESSL (Simulation Experiment Specification via a Scala Layer), and we employ $\text{MITL}_{[a,b]}$ (Metric Interval Temporal Logic) and CSL (Continuous Stochastic Logic) to specify the expected model behavioral properties, against which the simulation results are checked. Since we are experimenting with stochastic models, to alleviate the influence of stochasticity on the result checking, the LOESS (LOcal regrESSion) method is used to smooth simulation trajectories, which contain stochastic noise.

By bringing together a declarative, unambiguous specification of simulation experiments, statistical model checking, and certain adaptation rules, simulation experiments performed with previous versions of a model can be used to automatically check properties after extending a model and, thereby, illuminate the impact of these changes on model behavior. By employing this approach during model development, the intricate process of successively refining, extending and validating models and the process' documentation are facilitated.

With the promising results of applying our approach in a biological case study, i.e., developing three models of a receptor ligand pathway, we have shown the benefit of the presented approach in providing assistance for model development.

## Acknowledgments

This work was partly supported by the CSC (China Scholarship Council) and the National Natural Science Foundation of China under grants No. 61374185 and No.61402486, the German Research Foundation (DFG) via research grant UH-66/15-1 and research grant UH-66/14.

## References

[1] R. G. Sargent, Verification and validation of simulation models, Journal of simulation 7 (1) (2013) 12–24.
[2] R. Ewald, A. M. Uhrmacher, SESSL: A domain-specific language for simulation experiments, ACM Transactions on Modeling and Computer Simulation 24 (2) (2014) 11:1–11:25.
[3] A. Legay, B. Delahaye, S. Bensalem, Statistical model checking: An overview, in: Runtime Verification, 2010, pp. 122–135.
[4] D. Waltemath, R. Adams, D. A. Beard, F. T. Bergmann, U. S. Bhalla, R. Britten, V. Chelliah, M. T. Cooling, J. Cooper, E. J. Crampin, et al., Minimum information about a simulation experiment (MIASE), PLoS computational biology 7 (4) (2011) e1001122.
[5] H. Rahmandad, J. D. Sterman, Reporting guidelines for simulation-based research in social sciences, System Dynamics Review 28 (4) (2012) 396–411.
[6] J. Hillston, A tool to enhance model exploitation, Performance evaluation 22 (1) (1995) 59–74.
[7] C. U. Smith, C. M. Lladó, R. Puigjaner, Model interchange format specifications for experiments, output and results, The Computer Journal 54 (5) (2011) 674–690.
[8] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, et al., The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud, Nucleic acids research (2013) gkt328.
[9] J. Ribault, G. Wainer, Using workflows and web services to manage simulation studies (wip), in: Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium, Society for Computer Simulation International, 2012, p. 50.
[10] J. Schützel, D. Peng, A. M. Uhrmacher, L. F. Perrone, Perspectives on Languages for Specifying Simulation Experiments, in: Proceedings of the 2014 Winter Simulation Conference, WSC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 2836–2847.
[11] D. Ghosh, DSLs in Action, Manning Publications Co., Greenwich, CT, USA, 2011.
[12] D. Waltemath, R. Adams, F. T. Bergmann, et al., Reproducible computational biology experiments with SED-ML - the simulation experiment description markup language, BMC systems biology 5 (1) (2011) 198.
[13] R. R. Adams, SED-ED, a workflow editor for computational biology experiments written in SED-ML, Bioinformatics 28 (8) (2012) 1180–1181.
[14] A. W. Hallagan, The design of XML-based model and experiment description languages for network simulation, honors Thesis, Bucknell University (2010).
[15] J. Himmelspach, A. M. Uhrmacher, Plug'N Simulate, in: Proceedings of the 40th Annual Simulation Symposium, ANSS '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 137–143.
[16] D. Peng, R. Ewald, A. M. Uhrmacher, Towards Semantic Model Composition via Experiments, in: Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation, ACM, New York, NY, USA, 2014, pp. 151–162.
[17] C. Maus, S. Rybacki, A. M. Uhrmacher, Rule-based multi-level modeling of cell biological systems, BMC Systems Biology 5 (1) (2011) 166.
[18] T. Warnke, T. Helms, A. M. Uhrmacher, Syntax and semantics of a multi-level modeling language, in: Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM PADS '15, ACM, New York, NY, USA, 2015, pp. 133–144.
[19] M. Courtot, N. Juty, Knüpfer, et al., Controlled vocabularies and semantics in systems biology, Molecular systems biology 7 (1) (2011) 543.
[20] J. O. Dada, I. Spasić, N. W. Paton, P. Mendes, SBRML: a markup language for associating systems biology data withmodels, Bioinformatics 26 (7) (2010) 932–938.
[21] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT Press, Cambridge, Mass. [u.a.], 1999.

[22] M. Carrillo, P. A. Góngora, D. A. Rosenblueth, An overview of existing modeling tools making use of model checking in the analysis of biochemical networks, Frontiers in plant science 3 (2012) 155.
[23] F. Ciesinski, M. Größer, On probabilistic computation tree logic, in: C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, M. Siegle (Eds.), Validation of Stochastic Systems, Sp, Berlin Heidelberg, 2004, pp. 147–188.
[24] K. Sen, M. Viswanathan, G. Agha, On Statistical Model Checking of Stochastic Systems, in: Computer Aided Verification, Vol. 3576 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2005, pp. 266–280.
[25] F. Fages, A. Rizk, On temporal logic constraint solving for analyzing numerical data time series, Theoretical Computer Science 408 (1) (2008) 55–65.
[26] R. Alur, T. Feder, T. A. Henzinger, The Benefits of Relaxing Punctuality, J. ACM 43 (1) (1996) 116–146.
[27] E. M. Clarke, J. R. Faeder, C. Langmead, et al., Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway, in: Computational Methods in Systems Biology, Vol. 5307 of Lecture Notes in Computer Science, Sp, Berlin Heidelberg, 2008, pp. 231–250.
[28] O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Springer, Heidelberg, 2004, pp. 152–166.
[29] Y. Shoham, Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence., Tech. rep., Yale Univ., New Haven, CT (USA) (May 1987).
[30] J. Ma, B. Knight, A reified temporal logic, The Computer Journal 39 (9) (1996) 800–807.
[31] F. Bacchus, J. Tenenberg, J. A. Koomen, A non-reified temporal logic, Artificial Intelligence 52 (1991) 87–108.
[32] M. Grohe, Generalized Model-Checking Problems for First-Order Logic, in: Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science, STACS '01, Springer-Verlag, London, UK, UK, 2001, pp. 12–26.
[33] Wikipedia, Local regression, `https://en.wikipedia.org/wiki/Local_regression`, accessed 24 February 2016 (2016).
[34] W. S. Cleveland, Robust locally weighted regression and smoothing scatterplots, Journal of the American statistical association 74 (368) (1979) 829–836.
[35] Commons Math Developers, Apache commons math, release 3.6, Available from `https://commons.apache.org/proper/commons-math/download_math.cgi` (2016).
[36] H. L. S. Younes, R. G. Simmons, Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling, in: E. Brinksma, K. G. Larsen (Eds.), Computer Aided Verification, no. 2404 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002, pp. 223–235.
[37] T. Mens, K. Czarnecki, P. V. Gorp, 04101 discussion–a taxonomy of model transformations, in: Dagstuhl Seminar Proceedings, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2005.
[38] D. Waltemath, R. Henkel, R. Hälke, M. Scharm, O. Wolkenhauer, Improving the reuse of computational models through version control, Bioinformatics 29 (6) (2013) 742–748.
[39] D. A. Lauffenburger, J. J. Linderman, Receptors: models for binding, trafficking, and signaling, Oxford University Press, New York, 1996.
[40] T. Helms, C. Maus, F. Haack, A. M. Uhrmacher, Multi-level modeling and simulation of cell biological systems with ml-rules: A tutorial, in: Proceedings of the 2014 Winter Simulation Conference, WSC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 177–191.
[41] T. Helms, M. Luboschik, H. Schumann, A. Uhrmacher, An approximate execution of rule-based multi-level models, in: A. Gupta, T. Henzinger (Eds.), Computational Methods in Systems Biology, Vol. 8130 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2013, pp. 19–32.
[42] T. Helms, R. Ewald, S. Rybacki, A. M. Uhrmacher, Automatic runtime adaptation for component-based simulation algorithms, Acm Transactions on Modeling & Computer Simulation 26 (1) (2015) 1–24.
[43] K. H. Mayo, M. Nunez, C. Burke, C. Starbuck, D. Lauffenburger, C. R. Savage, Epidermal growth factor receptor binding is not a simple one-step process., Journal of Biological Chemistry 264 (30) (1989) 17838–44.
[44] E. De Maria, F. Fages, S. Soliman, On coupling models using model-checking: Effects of irinotecan injections on the mammalian cell cycle, in: P. Degano, R. Gorrieri (Eds.), Computational Methods in Systems Biology, Vol. 5688 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin Heidelberg, 2009, pp. 142–157.
[45] O. Pârvu, D. Gilbert, Automatic validation of computational models using pseudo-3d spatio-temporal

model checking, BMC Systems Biology 8 (1) (2014) 1–24.
[46] K. Thobe, A. Streck, H. Klarner, H. Siebert, Model integration and crosstalk analysis of logical regulatory networks, in: P. Mendes, J. Dada, K. Smallbone (Eds.), Computational Methods in Systems Biology, Vol. 8859 of Lecture Notes in Computer Science, Springer International Publishing, Berlin Heidelberg, 2014, pp. 32–44.
[47] F. Cellier, Continuous System Modeling, Continuous System Modeling, Springer, New York, 1991.
[48] R. Henkel, O. Wolkenhauer, D. Waltemath, Combining computational models, semantic annotations and simulation experiments in a graph database, Database 2015 (2015) bau130.
[49] J. Cooper, M. Scharm, G. R. Mirams, The cardiac electrophysiology web lab, Biophysical journal 110 (2) (2016) 292–300.
[50] H. K. Leung, L. White, Insights into regression testing [software testing], in: Proceedings of the Conference on Software Maintenancen, IEEE, 1989, pp. 60–69.
[51] E. Engström, P. Runeson, M. Skoglund, A systematic review on regression test selection techniques, Information and Software Technology 52 (1) (2010) 14–30.
[52] S. Yoo, M. Harman, Regression Testing Minimization, Selection and Prioritization: A Survey, Softw. Test. Verif. Reliab. 22 (2) (2012) 67–120.
[53] P. Ballarini, M. L. Guerriero, Query-based verification of qualitative trends and oscillations in biochemical systems, Theoretical Computer Science 411 (20) (2010) 2019–2036.
[54] R. B. Cleveland, W. S. Cleveland, I. Terpenning, STL: A Seasonal-Trend Decomposition Procedure Based on Loess, Journal of Official Statistics 6 (1) (1990) 3.
[55] M. Češka, D. Šafránek, S. Dražan, L. Brim, Robustness analysis of stochastic biochemical systems, PLoS ONE 9 (4) (2014) e94553.
[56] A. Rizk, G. Batt, F. Fages, S. Soliman, A general computational method for robustness analysis with applications to synthetic gene networks, Bioinformatics 25 (12) (2009) i169–i178.