Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

GENERATING SIMULATION EXPERIMENTS BASED ON MODEL DOCUMENTATIONS AND TEMPLATES

Andreas Ruscheinski
Kai Budde
Tom Warnke
Pia Wilsdorf
Bjarne C. Hiller
Marcus Dombrowsky
Adelinde M. Uhrmacher

Institute of Computer Science
University of Rostock
Albert-Einstein-Straße 22
Rostock, 18059, GERMANY

ABSTRACT

An increasing number of approaches for specifying and executing simulation experiments emphasizes the desire to make this part of modeling and simulation studies explicit, and thus, also easier to replicate. We take this one step further by automatically generating simulation experiment specifications from documentations. Based on a template-based approach and documentations, we show how simulation experiment specifications can be generated and executed for experiments, such as statistical model checking and sensitivity analysis, and we identify crucial challenges.

1 INTRODUCTION

The value of making simulation models accessible and reusable is undisputed in modeling and simulation. This has been documented in guidelines on how to describe simulation models, such as Minimum Information Required in the Annotation of Models (MIRIAM) (Le Novère et al. 2005) and Preferred Model Reporting Requirement (PMRR) (Rahmandad and Sterman 2012) or more recently (Monks et al. 2018), in standards for specifying and describing simulation models, such as the Systems Biology Markup Language (SBML) (Hucka et al. 2003) and the ODD (Overview, Design concepts, and Details) protocol (Grimm et al. 2010), and finally in model libraries, such as the BioModels Database (Le Novère et al. 2006) or the library of agent-based simulation models (Rollins et al. 2014), which rely on or at least encourage the use of the respective standards.

Despite the existence of these documentation guidelines, standardized formats or protocols, and simulation model libraries, only a fraction of modeling and simulation researchers is investing the effort to thoroughly document simulation models. For instance, a recent survey has shown that only 7 % of published agent-based models have used the ODD protocol (Janssen 2017). Also in the model repository for agent-based simulation models, OpenABM, only a fraction of papers provides a detailed and thorough ODD description. Even if standards are used, often crucial, complementary information is missing. For example, a search for the simulation experiment standard “SED-ML” (Köhn and Le Novère 2008) in the BioModels Database returns only a few hits out of more than 500 curated models encoded in the SBML standard (as of May 2018). Information about simulation experiments, however, does not only support reproducibility of the published simulation results, but also supports the credibility of the simulation model.

Therefore, simulation experiments should be part of model annotations, as also requested by the ODD protocol and in reporting guidelines.

This continuing lack of or incompleteness in documenting simulation models can be addressed in several ways. One is to decrease the user's efforts required for thorough documentation (Bergmann et al. 2014; Ruscheinski and Uhrmacher 2017). Another possibility is to incentivize authors to invest efforts in documentation by exploiting the documentation to ease other tasks in model development. The approach we are presenting caters to both objectives by facilitating a thorough documentation via generating declarative simulation experiment specifications and by incentivizing the user to document via exploiting the documentation during model development for generating simulation experiments (semi-)automatically.

2 TEMPLATE-BASED SIMULATION EXPERIMENT GENERATION

A declarative specification of experiments facilitates the manipulation (and finally generation) of simulation experiments and, due to their readability, simulation experiments can directly serve as documentation. As various types of simulation experiments, such as optimization, sensitivity analysis, or statistical model checking, are used while developing a model, our approach relies on blueprints for these simulation experiment types, which are adapted to the concrete simulation model and study based on information provided in the documentation of simulation models. We pursue a template-based approach in which the blueprints are defined as templates to be processed by a template engine. In addition to templates, the template engine needs a data model to generate executable simulation experiment specifications.

The simulation experiment templates are specified in a template language and contain immutable sections describing the basic structure of the simulation experiment type in the target language as well as mutable sections (i.e., template variables that serve as placeholders to be filled by the template engine). Templates are defined according to the syntax of a target experiment specification language, which is the Simulation Experiment Specification on a Scala Layer (SESSL) (Ewald and Uhrmacher 2014) language in our case. To complete these experiment templates, a vocabulary is introduced that specifies the expected content and, optionally, default values for each template variable. A data model is created by identifying the required information from the model documentation, transforming it into an expression in the target experiment specification language, and relating it to specific template variables. Finally, based on the data model and the template, the template engine generates the experiment specifications as output.

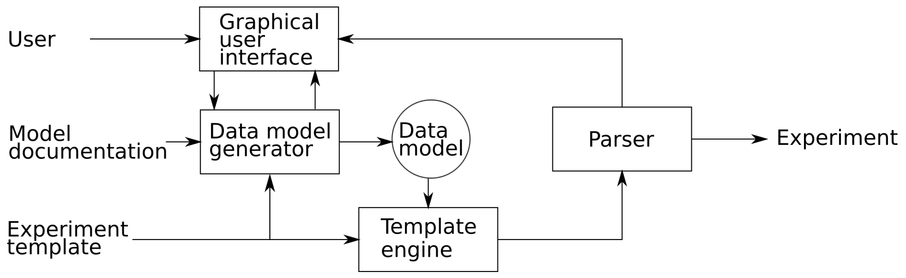

Figure 1: Overview of the experiment generation process.

We have chosen a semi-automatic approach for completing the experiment templates (https://git.informatik.uni-rostock.de/mosi/exp-generation; Accessed Aug. 22, 2018). Thus, the user can check and correct the information obtained from the model documentation and, if necessary, complete the data model with missing information. The experiment generation process is illustrated in Figure 1. The user loads the model documentation and the experiment template. Then, the data model generator uses the experiment template and the model documentation to generate a data model. Here, the template variables of the experiment template guide the information identification process of the data model generator. The resulting data model, a map of variables to expressions, contains the information necessary to fill in the template and generate the experiment. The graphical interface enables the user to check, modify, and complete this derived data model. We use the default values defined in the vocabulary (e.g., the default simulator for the simulation model or a test method for statistical model checking experiments) to minimize the required user input. The resulting completed data model together with the template serve as input for the template engine to generate the experiment specification. To prevent the generation of invalid experiment specifications due to incorrect user input, we implement a syntax check using a parser for our target specification language SESSL. The parser checks the generated experiment specification and reports errors to the graphical user interface. Only if the syntactical check succeeds, the generated experiment specification is shown to the user.

The main challenge during this process is to extract the required information for the template variables from the model documentation. In particular, the data model generator has to support different forms of model documentations a simulation model might be annotated with (e.g., verbal narratives, tabular representations, or formal languages), where each type requires a different method to identify the information. The data model generator works independently of a target language since it only needs a defined vocabulary and thus is reusable for other target simulation experiment languages. However, the syntax of expressions for identified information of template variables as well as the templates themselves need to be adapted for the different target languages, their supported formalisms, methods, and integrated tools.

Next, we discuss the challenges the data model generator faces when generating pairs of template variables and expressions based on a vocabulary and different documentation types. Thereafter, we demonstrate how templates are defined for a target specification language using two different experiment types.

3 DATA MODEL GENERATOR: MAPPING DATA TO TEMPLATE VARIABLES

The template engine FreeMarker (FreeMarker 2018) expects a map of template variable to data as an input. This map is generated by the data model generator. The generator's work depends on the template to be filled (i.e., the kind of simulation experiment that determines the template's variables), which needs to be mapped to concrete data. These variables comprise typical constituents of a simulation experiment specification, such as the reference to the simulation model or the simulation algorithm and, in addition, cover experiment type specific information, such as the formal behavior specification for a statistical model checking experiment or the experiment design for a sensitivity analysis. This information needs to be identified either by the user interactively or by automatically extracting the information from the documentation of the simulation model. To do so, the different documentation formats provide different challenges for the methods to be exploited. The more formal and unambiguous the description, also in relation to the implemented simulation model, the more likely the automatic procedure will succeed.

In the following, we first briefly discuss the vocabulary for the two simulation experiment types: statistical model checking and sensitivity analysis. Thereafter, we look at different documentation formats for simulation models and discuss possibilities as well as requirements for extracting the required information.

3.1 Vocabulary - Template Variables

Simulation experiments allow generating data from a simulation model by exerting it through its inputs (Cellier and Greifeneder 2006, p. 4). Constituents of a simulation experiment specification are a reference to the model, the initial model state, parameter values (also called model configuration), a reference to the simulation algorithm to execute the model, means to observe the generated data, means to analyze the data, the stop condition, and means to steer the generation of new model configurations (possibly based on the analysis results so far). In addition, if the simulation model is stochastic, means to generate random numbers, means to determine a suitable number of replications, and means to analyze the replications are needed. Note that some template variables expect single values, for instance ${model_path} or ${simulator}, and others are specified as blocks (e.g., "varBlock" or "obsBlock" are corresponding to a lists of pairs that are iterated by the template engine).

This basic information needs to be complemented for specific simulation experiments and the vocabulary needs to be refined accordingly. (For the information required for a basic simulation experiment see upper

Table 1: Overview of the template variable vocabulary for basic simulation experiment, statistical model checking experiments and sensitivity analysis experiments.

Template Variable	Description	Default value
${model_path}	Path to the simulation model	–
varBlock	Variable-value pairs for the model configuration	–
${simulator}	Simulator used for the simulation	StandardSimulator()
${stop_time}	Stop time of the simulation	–
${replications}	Number of replications for the experiment	–
${observe_at}	Observation times points	–
obsBlock	Variable-expression pairs for the observations	–
${test_method}	Method to test property	SequentialProbabilityRatioTest
${test_parameter}	Parameters for test method	p=0.8
${property_type}	Type of the property	MITL
${property_exp}	Expression for the property	–
${vresult_function}	Function for the validation result	print(result)
baseBlock	Variable-value pairs for the base case	–
sensBlock	Variable-value pairs for the sensitivity case	–
${aresult_function}	Function sensitivity analysis result	print(result)

part of Table 1.) One experiment type that gained interest over the last two decades is model checking, which subsumes methods to test whether a model satisfies a formally specified property usually expressed in temporal logics (e.g., MITL (Maler and Nickovic 2004)). Statistical model checking is based on simulations and determines whether a random simulation run produced by the model will show the specified behavior with at least some specified probability p (Agha and Palmskog 2018). The middle part of Table 1 lists the vocabulary needed for a statistical model checking experiment: the formal language in which properties will be defined, the properties themselves, the statistical model checking method that shall be applied on the trajectories, and the probability p with which the properties should hold.

In the case of sensitivity analysis, the goal is to determine how strongly simulation model outputs are affected by changes in the model parameters (Kleijnen 1995). Sensitivity analysis can be conducted locally or globally. Local sensitivity analysis determines how *small* changes around a specific model configuration affect the model outputs, whereas global sensitivity analysis considers a large parameter space to determine "how the uncertainty in the output [...] can be apportioned to different sources of uncertainty in the model input" (Saltelli et al. 2004, p. 45). One approach for local sensitivity analysis is to vary the model parameters following an experiment design around a base configuration, observe the simulation outputs, and calculate the effects of parameter changes on the simulation output (Kleijnen 1995). Thereby, the experiment design only scans few values for each parameter. In commonly used designs for local sensitivity analysis, such as the 2^k design, two values for each parameter are needed (lower part of Table 1).

To support the generation of mappings for other experiment types, such as simulation-based optimization (Amaran et al. 2014), the vocabulary has to be extended to reflect the specific needs. Similarly, if other methods shall be used, for instance a different test for statistical model checking, alternative parameters might be required. Certain tools might provide default values for some of the variables, for all others, the information needs to be extracted automatically from the data or manually completed by the user.

3.2 Documentation

To extract the data to be mapped to a specific variable, the accessibility and unambiguity of the documentation is crucial. The less formal the representation is, the less information can be extracted (automatically). Documentation formats range from verbal narratives to formal languages. However, in most cases a mixture of different formats is used that may include links to external data sources or ontologies.

Verbal Narratives: The most common and, unfortunately for extracting, worst case are verbal narratives even if they come in a structured form as suggested by the ODD protocol (Grimm et al.

2010). For example in Klabunde et al. (2015), a detailed ODD description of a migration model is given. Among other information, ODD expects information about model variables and how they are observed. The corresponding paragraph "Observation" contains the information to be mapped to the template variables ${obsBlock["obs_var"]} (the variables in which the observed information is to be recorded) and ${obsBlock["obs_exp"]} (the expression defining which part of the simulation model to observe and when). It is stated that "many other data items were measured during model design and testing, such as (...) distribution of number of children across women" (Klabunde et al. 2015, p.18). The transformation of the ODD description into an executable observation reads as observeAt(range(1982, 1, 2050)), observe("children" $\sim$ expressionDistribution("Person", "ego.children.size()")). Such a transformation is clearly tool-specific. Thus, without knowledge about the concrete observation features and syntax that the simulation or experimentation tool provides (in the above case SESSL and ML3 (Warnke et al. 2017)) and without sophisticated text mining capabilities, observation template variables cannot be filled.

Tabular representations: Publications about practical simulation studies, such as (Haack et al. 2015), often contain tables that summarize the parameters, variables, their respective values, meaning, and origin–in short information about the model configuration. Also ODD requests tabular representations to list parameters and their values as well as variables with their initial values and origin. If tabular listings use consistent names and contain correct values, they can be used to fill in the template variables (i.e., to fill template blocks by simply extracting name-value pairs). For example, parameters from Haack et al. (2015) could be extracted into the following block structure expressing inital molecule numbers for the WNT/β-catenin model: [(WNT,220),(LRP6(mem),4000),(CK1y(mem),5000),...]. However, again too few constraints on how the information should be presented prevent an automatic mapping. For example, if we find only verbal narratives in these tables and not the concrete variable names, an automatic mapping without additional efforts is not possible. In addition, the use of inappropriate data exchange formats hampers the extraction of important information (e.g., if lists of values are presented as PDF documents rather than as CSV files).

Formal languages: If the model documentation contains some expression in a formal language, this expression can be extracted and used comparatively easily. For example, a model might be annotated with a specification of its behavior in a temporal logic. The concrete logic formalism can be used to populate the template variable ${property_type} and the actual property can become the input for the template variable ${property_exp}. Similarly, the model documentation might contain mathematical expressions, which can then be extracted and exploited to populate template variables, for example, by deriving ${test_parameter} from an expression such as $p \geq 0.8$. In any case, if expressions shall be extracted automatically from the documentation, the language in which these expressions are defined has to be either directly accessible within the experimentation tool or some translation of the expressions will be necessary to adapt them to the concrete syntax of the experimentation tool. However, writing or understanding expressions in a formal language may be tedious. Therefore, for the expressions to be intelligible the model documentation should also include a verbal explanation to serve as a (simplified) documentation for the user.

Data sources: Simulation models are typically developed based on real data. Publications usually contain this information directly in the form of tables (see above). However, sometimes the data is included as supplementary material and often the reader is redirected to another source, thus the required data to fill the template variables is not directly accessible. For example, in the above ODD description the data collected within the MAFE project (INED. 2018) is mentioned as a central data source, however, currently the data sets can only be accessed by browsing through the respective databases. With the emerging scientific culture to publish data sets with Digital Object Identifiers (DOIs) (ISO. 2018), this is about to change.

Ontologies: Ontologies represent a broad spectrum of knowledge for various application domains and thus can play different roles in the documentation of simulation models and the generation of simulation experiments. On the one hand, they are used to denote the meaning behind variables, parameters, and reactions in the real world. For example, there are general ontologies that capture concepts and terminology commonly used within a domain but also ontologies that are rather specific to a certain subdomain (e.g., d'Aquin and Noy (2012) list ontology libraries where ontologies from various scientific fields can be

```
execute{
   new Experiment with Observation with StatisticalModelChecking {
      model = "${model_path}" //path to the simulation model

      <#list varBlock as configuration> //directive to iterate over the variable-value pairs to specify
          the model configuration
         set("${configuration[0]}" <~ ${configuration[1]})
      </#list>

      simulator = ${simulator} //simulator used for the simulation
      stopTime = ${stop_time} //stop time of the simulation

      observeAt(${observe_at}) //observation time points
      <#list obsBlock as observations> //directive to iterate the tuples generated from the observation
          variables and their filter expression to specify what should be observed
         observe("${observations[0]}" ~ ${observations[1]})
      </#list>

      test = ${test_method}(${test_parameter}) //the test method with their parameters
      prop = ${property_type}(${property_exp}) //property type and the expression for the property
      withCheckResult {result => ${vresult_function}} //function executed with the validation result
   }
}
```

Listing 1: Statistical model checking experiment template for SESSL.

found). The information these ontologies provide supports the interpretation of the simulation model as well as the achieved results. On the other hand, some ontologies also refer to methods used within the experimentation process, which can be used to select the right methods from the available ones in a concrete setting. For example, ontologies may contain information about existing simulation algorithms, describe types of software, or refer to the specified behavioral properties (Bard and Rhee 2004). The extensive use of ontologies helps third-party reuse of simulation products and supports a more conceptual approach in documenting the simulation model. Moreover, initiatives such as the OBO Foundry (OBO Technical WG 2018) promote interoperability and re-use across ontologies by ensuring orthogonality of vocabularies and by introducing formal formats such as OWL or OBO (Bodenreider and Stevens 2006). In addition, W3C standards such as OWL bring Semantic Web features to other scientific fields, such as biology (e.g., techniques for linking and analysing data sets (Tirmizi et al. 2011)).

Next, we specify experiment templates for our target specification language SESSL for statistical model checking and sensitivity analysis experiments.

4 EXPERIMENT TEMPLATES

The experiment template presents a blueprint for a specific simulation experiment type and structures it into immutable and mutable sections of a target specification language. To demonstrate our approach, we specify experiment templates for statistical model checking and sensitivity analysis for the experiment specification language SESSL. SESSL is an internal domain-specific language that acts as a layer between user and simulation system and that supports a declarative description and direct execution of simulation experiments. SESSL experiments are characterized by *traits*, which provide features for experiment execution as well as configuration options for these features. Upon experiment execution, the configuration is translated into calls to an underlying simulation system.

To specify the experiments templates, we analyze pre-existing SESSL experiments to determine the immutable and mutable sections of simulation experiments. The resulting experiments templates are shown in Listing 1 and 2. Each experiment type requires the observation trait and uses the model, the model configuration, the simulation algorithm, the stop time, the observation times, and a specification of what to observe as configuration options. Therefore, we represent the observation trait and the option names as

```
 1 analyze((params, objective) => execute(
 2   new Experiment with Observation {
 3
 4    model = ... //basic setup shortened
 5
 6    replications = ${replications}
 7    observe(${observe_var} ~ ${observe_expression]})
 8
 9    for((name, value) <- params) set(name <~ value)
10
11    withReplicationsResult(result =>objective <~ result.mean("${observe_var}"))
12 })) using new TwoLevelFullFactorialSetup {
13   baseCase(
14    <#list baseBlock as configuration>
15        "${configuration[0]}" <~ ${configuration[1]},
16    </#list>
17   )
18   sensitivityCase(
19    <#list sensBlock as configuration>
20        "${configuration[0]}" <~ ${configuration[1]},
21    </#list>
22   )
23   withAnalysisResult(result => {${aresult_function}})
24 }
```

Listing 2: Sensitivity analysis experiment template for SESSL (basic experiment setup shortened).

the immutable sections in the experiment template and use the template variables to represent the option values as mutable sections. For example in SESSL, the path to the simulation model is configured by the model configuration option. The option name "model" is part of the immutable section of the template because each SESSL experiment requires a simulation model for its configuration, whereas the option value "${model_path}" depends on the concrete use case and therefore needs to be extracted by the data model generator to be set by the template engine (Listing 1, l. 3). Since we specify the model configuration in SESSL by multiple assignments, we have to generate these for our identified variable-value pairs in the "varBlock" variable of the data model. For this, we iterate over the identified pairs using the list directive (Listing 1, ll. 5–7). We use the same method to configure the observations. The basic experiment setup is shown in Listing 1 (ll. 3–15).

For statistical model checking experiments, SESSL requires an experiment type specific trait, the test method, the property that should be checked, and a result function as configuration options. As before, we add the trait and option names to the immutable sections whereas the option values are represented by template variables (Listing 1, l. 2 and ll. 17–19). To configure the test method and the property, we use two template variables for each to separate the test method from its parameterization and to separate the test property type from its specification as well (Listing 1, l. 2 and ll. 17–18).

The current version of SESSL only supports the two-level-full-factorial method for the local sensitivity analysis, thus the experiment template is tailored toward this method. For this, we have to configure the base and the sensitivity case for the experiment design and a function executed with the experiment result. As before, we represent these option names as immutable sections whereas the option values are represented by template variables (Listing 2, ll. 12–23). Since the base and the sensitivity case are specified as model configuration, we iterate over the corresponding blocks as before (Listing 2, ll. 13–17 and 18–22).

5 THE APPROACH AT WORK

We demonstrate our approach by using a simulation model from cell biology realized in ML-Rules (Maus et al. 2011) and annotated with tables that contain information about parameters and state variables as well as formally defined behavioral hypotheses.

5.1 Simulation Model and Documentation

The model used in our case study shows kinetics of the binding of EGF to cellular transmembrane receptors of human foreskin fibroblast cells (Mayo et al. 1989). In Peng et al. (2016), it served as a case study to show how simulation experiments can be reused in successively extending a model. Here, we use the model to illustrate how, based on documentations and simulation experiment templates, simulation experiments can be generated. The documentation of the model comprises two tables.

Table 2 contains all variables and parameters of the model, their values, units, uncertainties, sources and references. This information will be especially useful for sensitivity analysis. Table 3 contains expressions for hypotheses testing, which will be important for the model checking experiment. These tables are stored as CSV files, which is a typical data exchange format for tabular data, and bundled together with the simulation model.

Table 2: Variable and parameter table of the ternary complex model of EGF binding (* – experimentally determined, ** – values taken from (Peng et al. 2016; Mayo et al. 1989)).

Name	Type	Value	Units	Standard Deviation (in percent)	Source	Reference
nR	variable	6.0e4				(**)
nL	variable	1.05e9				(**)
nX	variable	2.4e4				(**)
kf	parameter	4.5e-13	1/s	16.7	(*)	(**)
kr	parameter	1.7e-2	1/s	53.3	(*)	(**)
ka	parameter	5.0e-9	1/s	83.3	(*)	(**)
ku	parameter	8.0e-5	1/s	35.7	(*)	(**)

Table 3: Hypotheses for validating the ternary complex model of EGF binding.

Observation Variable	Observation Expression	Description	Property Type	Property Expression
Complexes	count("C(count)")	Number of L-R-(X) Complexes above 1000 after 900 seconds	MITL	G(900,30000) (OutVar ("Complexes") > Constant(1000))
Complexes	count("C(count)")	Number of L-R-(X) Complexes above 3000 and below 4000 after 20000 seconds (approaches a steady state)	MITL	G(20000,30000) (Constant(3000) < OutVar("Complexes") and OutVar("Complexes") < Constant(4000))

5.2 Experiment Generation and Experiment Results

Experiments are generated by using the previously described semi-automatic approach for completing experiment templates as shown in Listings 1 and 2. The application starts with a graphical user interface where the user chooses an experiment template and imports the model documentation–in our case the CSV files representing the tables. After this, the data model generator generates a data model from the model documentations for the required template variables. The user can check and modify the identified information. The manual modifications of the extracted information are printed in bold in Listings 3 and 4.

5.2.1 Statistical Model Checking

Most information needed to complete the template can be extracted from the imported documentations (e.g., the observation variables and the expressions from Table 3). Missing details are being highlighted in the experiment specification and need to be added by the user. This includes the file path of the simulation model and the expression of the observation interval. Default values are available for the test parameters of the sequential probability ratio test and the result function. The complete experiment is shown in Listing 3.

```
execute {
  new Experiment with Observation with StatisticalModelChecking {
    model = "TernaryComplexModel.mlrj"

    simulator = SimpleSimulator()
    stopTime = 30001

    observeAt(range(0,1,30001))
    observe("Complexes" ~ count("C(count)"))

    test = SequentialProbabilityRatioTest(p = 0.8)
    prop = MITL(
      G(900,30000)(OutVar("Complexes")>Constant(1000)) and}
      G(20000,30000)(OutVar("Complexes")>Constant(3000) and OutVar("Complexes")<Constant(4000)))
    withCheckResult { result => println(result)}
  }
```

Listing 3: Statistical model checking experiment for SESSL (bold: inserted by the user).

The statistical model checking experiment has been executed in parallel using 23 cores. 46 repetitions were needed to determine that the MITL expression is true for the given parameters. We conclude that our simulation model behaves qualitatively and quantitatively very similar to experimental wet-lab results of (Mayo et al. 1989).

5.2.2 Sensitivity Analysis

We consider a two-level-full-factorial (2^k) design with 100 repetitions for three parameters and use, as an example for our sensitivity case, a 10 % increase of the parameter values. Our choice of parameters includes the kinetic production rate of the ternary complex, *ka*, which shows the highest uncertainty. The second parameter, *kr*, describes the fast destruction process of the binary (EGF-receptor) complex. Finally, the kinetic production rate of the binary complex, kf, is regarded because of the necessary conversion from a concentration based rate in $M^{-1}s^{-1}$ to a number based rate in s^{-1}. The values for the base and sensitivity cases as well as the results of the executed experiment are shown in Table 4. The mean $\pm$ standard deviation of the number of ternary complexes after a simulation time of 30001 s is 3548 ± 54. A 10 % increase of *ka* and kf leads to an increase of 187 and 320 or 5 % and 9 %, respectively. An increase of the destruction rate *kr* results in a decrease of 326 or 9 %. The outcome of this 2^k analysis was expected because a higher production rate should increase the number of complexes and a higher destruction rate should lower that number. Interestingly, the interaction effects are negative and rather small. Therefore, we can conclude that the individual effects are slightly dampened by interactions of the parameters.

Table 4: Input (parameter name, base and sensitivity case values)z and output (effects regarding the change of the number of ternary complexes after 30001 s of simulation time) of sensitivity analysis. The individual effect is the difference of the base case and the sensitivity case where one parameter value was changed. The total effect is the sum of all interaction effects for all design points where the parameter under consideration is equal to the sensitivity case. The difference of the total effect and the individual effect results in the total interaction effect.

Parameter name	**Base case value**	**Sensitivity case value**	**Individual effect**	**Total effect**	**Total interaction effect**
ka	5e-9 s^{-1}	5.5e-9 s^{-1}	187	185	-2
kr	1.7e-2 s^{-1}	1.87e-2 s^{-1}	-287	-326	-39
kf	4.5e-13 s^{-1}	4.95e-13 s^{-1}	331	320	-11

```
analyze((params, objective) => execute(
new Experiment with Observation {
  model = "TernaryComplexModel.mlrj"

  simulator = SimpleSimulator()
  replications = 100
  stopTime = 30001

  observeAt(30001)
  observe("Complexes" ~ count("C(count)"))

  for((name, value) <- params)
  set(name <~ value)

  withReplicationsResult(result =>
  objective <~ result.mean("Complexes")
  )
})) using new TwoLevelFullFactorialSetup {
  baseCase(
  "ka" <~ 5e-9, "kr" <~ 1.7e-2, "kf" <~ 4.5e-13
  )
  sensitivityCase(
  "ka" <~ 5.5e-9, "kr" <~ 1.87e-2, "kf" <~ 4.95e-13
  )
  withAnalysisResult(result => {
  result.printCompleteReport()
})
}
```

Listing 4: Sensitivity analysis experiment for SESSL (bold: inserted by the user).

6 CONCLUSION

A variety of simulation experiments are typically executed during developing a simulation model. Our approach supports the automatic generation of simulation experiments and thus facilitates designing and executing simulation experiments. Therefore, we exploit predefined templates for diverse simulation experiment types and the documentation of a simulation model. For the approach to work, the latter needs to present and to structure information often found in publications in a manner that allows an unambiguous interpretation. We illustrate this by behavior hypotheses, not only verbally described but also formally specified in a temporal logic and by parameter tables which, in addition to descriptions and sources, include the concrete parameter names, values, and variations. We use this information to automatically generate and execute statistical model checking experiments and sensitivity analysis experiments respectively. Our approach relies on, but also promotes a more rigorous documentation of simulation models and thus also, due to making simulation experiments explicit, contributes to the reproducibility of modeling and simulation research. In addition, the automatic generation and execution of simulation experiments based on the documentation keep simulation models and its documentation consistent.

ACKNOWLEDGMENTS

The research was funded by the DFG UH 66/18 ’GrEASE’ and by DFG CRC 1270 ’Elaine’.

REFERENCES

Agha, G., and K. Palmskog. 2018. “A Survey of Statistical Model Checking”. *ACM Transactions on Modeling and Computer Simulation* 28(1):6:1–6:39.

Amaran, S., N. V. Sahinidis, B. Sharda, and S. J. Bury. 2014. “Simulation Optimization: A Review of Algorithms and Applications”. *4OR* 12(4):301–333.

Bard, J. B., and S. Y. Rhee. 2004. “Ontologies in Biology: Design, Applications and Future Challenges”. *Nature Reviews Genetics* 5(3):213.

Bergmann, F. T., R. Adams, S. Moodie, J. Cooper, M. Glont, M. Golebiewski, M. Hucka, C. Laibe, A. K. Miller, D. P. Nickerson, B. G. Olivier, N. Rodriguez, H. M. Sauro et al. 2014. “COMBINE Archive and OMEX Format: One File to Share all Information to Reproduce a Modeling Project”. *BMC bioinformatics* 15(1):369.

Bodenreider, O., and R. Stevens. 2006. “Bio-ontologies: Current Trends and Future Directions”. *Briefings in Bioinformatics* 7(3):256–274.

Cellier, F. E., and J. Greifeneder. 2006. *Continuous System Modeling*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

d’Aquin, M., and N. F. Noy. 2012. “Where to publish and find ontologies? A survey of ontology libraries”. *Web Semantics: Science, Services and Agents on the World Wide Web* 11:96 – 111.

Ewald, R., and A. M. Uhrmacher. 2014. “SESSL: A Domain-specific Language for Simulation Experiments”. *ACM Transactions on Modeling and Computer Simulation* 24(2):11.

FreeMarker 2018. “Apache FreeMarker Manual for Freemarker 2.3.28”. https://freemarker.apache.org/docs/index.html. Accessed April 23, 2018.

Grimm, V., U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback. 2010. “The ODD Protocol: A Review and First Update”. *Ecological modelling* 221(23):2760–2768.

Haack, F., H. Lemcke, R. Ewald, T. Rharass, and A. M. Uhrmacher. 2015. “Spatio-temporal Model of Endogenous ROS and Raft-Dependent WNT/Beta-Catenin Signaling Driving Cell Fate Commitment in Human Neural Progenitor Cells”. *PLOS Computational Biology* 11(3):1–28.

Hucka, M., A. Finney, H. Sauro, H. Bolouri, J. Doyle, H. Kitano, A. Arkin, B. Bornstein, D. Bray, A. Cornish-Bowden, A. Cuellar, S. Dronov, E. Gilles, M. Ginkel et al. 2003. “The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models”. *Bioinformatics* 19(4):524–531.

INED. 2018. “MAFE Project”. https://mafeproject.site.ined.fr/. Accessed May 7, 2018.

ISO. 2018. “Digital Object Identifier”. https://www.iso.org/obp/ui/#iso:std:iso:26324:ed-1:v1:en. Accessed May 7, 2018.

Janssen, M. A. 2017. “The Practice of Archiving Model Code of Agent-based Models”. *Journal of Artificial Societies and Social Simulation* 20(1):2.

Klabunde, A., F. J. Willekens, S. Zinn, and M. Leuchter. 2015. “An Agent-based Decision Model of Migration, Embedded in the Life course - Model Description in ODD+D Format”. MPIDR Working Papers WP-2015-002, Max Planck Institute for Demographic Research, Rostock, Germany.

Kleijnen, J. P. C. 1995. “Sensitivity Analysis and Optimization in Simulation: Design of Experiments and Case Studies”. In *Proceedings of the 1995 Winter Simulation Conference*, edited by C. Alexopoulos and K. Kang, 133–140. Piscataway, New Jersey: IEEE.

Köhn, D., and N. Le Novère. 2008. “SED-ML – An XML Format for the Implementation of the MIASE Guidelines”. In *Computational Methods in Systems Biology*, edited by M. Heiner and A. M. Uhrmacher, 176–190. Berlin, Heidelberg: Springer Berlin Heidelberg.

Le Novère, N., B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J. L. Snoep, and M. Hucka. 2006. “BioModels Database: A Free, Centralized Database of Curated, Published, Quantitative Kinetic Models of Biochemical and Cellular Systems”. *Nucleic acids research* 34(suppl_1):D689–D691.

Le Novère, N., A. Finney, M. Hucka, U. S. Bhalla, F. Campagne, J. Collado-Vides, E. J. Crampin, M. Halstead, E. Klipp, P. Mendes, P. Nielsen, H. Sauro, B. Shapiro, J. L. Snoep et al. 2005. “Minimum Information Requested in the Annotation of Biochemical Models (MIRIAM)”. *Nature biotechnology* 23(12):1509.

Maler, O., and D. Nickovic. 2004. “Monitoring Temporal Properties of Continuous Signals”. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, edited by Y. Lakhnech and S. Yovine, 152–166. Springer.

Maus, C., S. Rybacki, and A. M. Uhrmacher. 2011. “Rule-based Multi-level Modeling of Cell Biological Systems”. *BMC Systems Biology* 5(1):166.

Mayo, K. H., M. Nunezll, C. Starbuck, and D. Lauffenburger. 1989. “Epidermal Growth Factor Receptor Binding Is Not a Simple One- step Process”. *Journal of Biological Chemistry* 264(30):17838–17844.

Monks, T., C. S. M. Currie, B. S. Onggo, S. Robinson, M. Kunc, and S. J. E. Taylor. 2018. “Strengthening the Reporting of empirical simulation studies: Introducing the STRESS Guidelines”. *Journal of Simulation* 0(0):1–13.

OBO Technical WG 2018. “The OBO Foundry”. http://www.obofoundry.org. Accessed May 7, 2018.

Peng, D., T. Warnke, F. Haack, and A. M. Uhrmacher. 2016. “Reusing Simulation Experiment Specifications to Support Developing Models by Successive Extension”. *Simulation Modelling Practice and Theory* 68:33 – 53.

Rahmandad, H., and J. D. Sterman. 2012. “Reporting Guidelines for Simulation-based Research in Social Sciences”. *System Dynamics Review* 28(4):396–411.

Rollins, N. D., C. M. Barton, S. Bergin, M. A. Janssen, and A. Lee. 2014. “A Computational Model Library for Publishing Model Documentation and Code”. *Environmental Modelling & Software* 61:59–64.

Ruscheinski, A., and A. M. Uhrmacher. 2017. “Provenance in Modeling and Simulation Studies - Bridging Gaps”. In *Proceedings of the 2017 Winter Simulation Conference*, edited by A. D’Ambrogio, G. Zacharewicz, and N. Mustafee, 872–883. Piscataway, New Jersey: IEEE.

Saltelli, A., S. Tarantola, F. Campolongo, and M. Ratto. 2004. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Hoboken, NJ, USA: John Wiley & Sons.

Tirmizi, S. H., S. Aitken, D. A. Moreira, C. Mungall, J. Sequeda, N. H. Shah, and D. P. Miranker. 2011. “Mapping Between the OBO and OWL Ontology Languages”. *Journal of Biomedical Semantics* 2(1):S3.

Warnke, T., O. Reinhardt, A. Klabunde, F. Willekens, and A. M. Uhrmacher. 2017. “Modelling and Simulating Decision Processes of Linked Lives: An Approach Based on Concurrent Processes and Stochastic Race”. *Population studies* 71(sup1):69–83.

AUTHOR BIOGRAPHIES

ANDREAS RUSCHEINSKI is a Ph.D. student in the modeling and simulation group at the University of Rostock. His e-mail address is andreas.ruscheinski@uni-rostock.de

KAI BUDDE is a Ph.D. student in the modeling and simulation group at the University of Rostock. His e-mail address is kai.budde@uni-rostock.de

TOM WARNKE is a Ph.D. student in the modeling and simulation group at the University of Rostock. His e-mail address is tom.warnke@uni-rostock.de

PIA WILSDORF is a Ph.D. student in the modeling and simulation group at the University of Rostock. Her e-mail address is pia.wilsdorf@uni-rostock.de

BJARNE C. HILLER is a student assistant in the modeling and simulation group at the University of Rostock. His e-mail address is bjarne.hiller@uni-rostock.de

MARCUS DOMBROWSKY is a student assistant in the modeling and simulation group at the University of Rostock. His e-mail address is marcus.dombrowsky@uni-rostock.de

ADELINDE M. UHRMACHER is professor at the Institute of Computer Science, University of Rostock, and head of the modeling and simulation group. Her e-mail address is adelinde.uhrmacher@uni-rostock.de