

## **SIMULATION EXPERIMENT SCHEMAS – BEYOND TOOLS AND SIMULATION APPROACHES**

Pia Wilsdorf  
Marcus Dombrowsky  
Adelinde M. Uhrmacher

Julius Zimmermann  
Ursula van Rienen

Institute for Visual and Analytic Computing  
University of Rostock  
Albert-Einstein-Str. 22  
Rostock, 18059, GERMANY

Institute of General Electrical Engineering  
University of Rostock  
Albert-Einstein-Str. 2  
Rostock, 18059, GERMANY

### **ABSTRACT**

Simulation studies make use of various types of simulation experiments. For users, specifying these experiments is a demanding task since the specification depends on the experiment type and the idiosyncrasies of the used tools. Thus, we present an experiment generation procedure that guides users through the specification process, and abstracts away from the concrete execution environment. This abstraction is achieved by (1) developing schemas that define the properties of simulation experiments and dependencies between them, (2) specifying a mapping between schema properties and template fragments in the specification language of a target backend. We develop schema, template fragments, and mappings for stochastic discrete-event simulation, and show how the concepts can be transported to a different domain of modeling and simulation. Further, we expand the developed “simple” experiment schemas by a schema for experiment designs, and generate executable sensitivity analysis experiments, thereby demonstrating versatility and composability of our approach.

### **1 INTRODUCTION**

The last decade has seen an increasing interest in capturing simulation experiments explicitly, to facilitate their reuse and replication. The developed approaches include, e.g., model-based approaches such as (Teran-Somohano et al. 2015), or experiment specification via domain-specific languages like SESSL (Ewald and Uhrmacher 2014), or SED-ML (Waltemath et al. 2011). Despite these efforts for facilitating reuse and documentation of simulation experiments, specifying simulation experiments is still a challenge for modelers, since the specification relies on the idiosyncrasies of experiment types, used methods, and tools. To support the specification of simulation experiments, approaches like template-based experiment generation can be exploited, see (Ruscheinski et al. 2018). These templates encode knowledge about the diverse experiments, their methods, and the information needed, and thus form an essential part of the body of knowledge in modeling and simulation (Ören 2005). However, in the above case each template had to be tailored, not only to the experiment type, but in addition to the target specification language, thereby inhibiting versatility, and exchangeability of the approach.

Therefore, in this paper we extend the experiment generation process into a two-level process (see Section 2). At the first level, an abstract experiment specification is generated based on a modularized schema where each schema module defines the required and optional inputs for a common constituent of simulation experiment specifications, such as model initialization, simulator initialization, or observation and post-processing of results. At the second level, the abstract experiment specification is mapped to template fragments which, composed together, yield a concrete experiment specification in the specification

language of a selected modeling and simulation backend. In practice, this abstraction and modularization of simulation experiments will have numerous advantages, for example the modularization will allow for more flexible user support for specifying, extending, or composing simulation experiment specifications. Moreover, the clear decoupling of schema and implementation will allow for versatile use and reuse of the same (abstract) experiment specification for different tools without having to respecify the entire experiment. In addition, the schema modules essentially present guidelines for a straight-forward integration of new tools within the experiment generator, i.e., by simply implementing new schema-to-backend mappings.

To demonstrate our concepts we first develop schema, template fragments, and mappings for stochastic discrete-event simulation (DES) (Section 3). Thereafter, we show how the concepts can be transported to a different domain of modeling and simulation, i.e., the application of the finite element method (FEM) to solve electromagnetic problems (Section 4). Further, we extend the developed “simple” experiment schemas, which refer to the specification of single simulation runs, by a schema for sensitivity analysis. Using these extended schemas we generate executable sensitivity analysis experiments for two different modeling and simulation backends, i.e., SESSL/ML-Rules (Maus et al. 2011; Ewald and Uhrmacher 2014), and EMStimTools/YAML (Zimmermann 2019) (Section 5) to illustrate versatility and composability of our approach. Finally, we close the paper with a discussion of related work (Section 6), and conclusions including future work (Section 7).

## 2 EXPERIMENT GENERATION PROCESS

The extended experiment generation process is schematically depicted in Figure 1. There, four main tasks can be identified, i.e., input collection, schema validation, template rendering, and experiment execution. In the following these tasks and their respective inputs and outputs will be described, as well as how the tasks are intertwined in a rather iterative experiment generation process. The experiment generation process generates two experiment specifications at different levels of abstraction. The first one is an intermediate product called the *experiment object* which provides an abstract experiment specification in terms of the used methods and parameters. The second one, the final *experiment specification*, is a concrete experiment specification where the abstract methods and parameters have been mapped to implementations in a concrete modeling and simulation backend. The implementation of the experiment generation process is available under (Wilsdorf et al. 2019). This repository also contains full versions of the presented experiment schemas, experiment objects, and the resulting executable simulation experiment specifications.

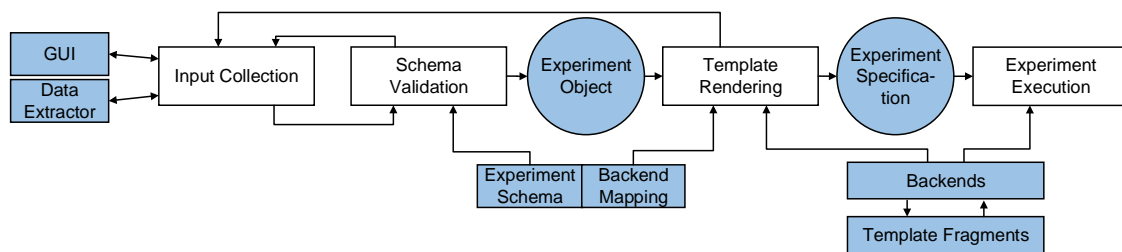


Figure 1: Flowchart illustrating the experiment generation process that handles simulation experiment specifications at two levels of abstraction, i.e., the implementation-agnostic experiment object, and the implementation-dependent experiment specification.

### 2.1 Input Collection

The first task in the experiment generation process is to collect all the necessary inputs to get an executable experiment specification, e.g., one has to enter information about the simulation model, the desired simulator, or what to observe. All inputs are collected in a name-value map, where the name corresponds to a specific

part of a simulation experiment specification (in the following called *property*), and the value is the collected input, which may be either a single value, or an array of values. In our realization we chose the multipurpose data interchange format JSON (JSON 2019) to store the collected name-value pairs. This map of values is passed to the schema validator (see Section 2.2). The schema validator checks the experiment structure, and the type of input values against a defined experiment schema. Moreover, from the schema the structure and behavior of a dynamic graphical user interface (GUI) can be derived (Teran-Somohano et al. 2015). This allows for an interactive, step-by-step input collection that guides the user through the specification process by requesting certain information, suggesting default values, and prompting error messages. The specification typically starts with a set of minimally required inputs, and presents these input fields in the GUI. Next, once an input for an input field is chosen, other conditionally required fields may be revealed. For example, once a method has been selected for statistical model checking, e.g., the sequential probability ratio test (Wald 1945), further inputs are required, i.e., the probability  $p$ , a (temporal) logic formula  $\phi$ , and optional error thresholds used to reject the null hypothesis. Thus, the experiment generation process is an iterative process where inputs are iteratively checked and requested.

For additional user support, this may be combined with a data extractor that automatically determines inputs for certain schema properties by deriving them from other sources (Ruscheinski et al. 2018), e.g., a suitable model configuration may be found by extracting relevant parameter names from the simulation model, and corresponding values from model documentations such as parameter tables.

## 2.2 Schema Validation

The schema validation task checks the collected input data against a defined experiment schema, e.g., using JSON Schema (JSON Schema 2019). Experiment schemas represent the basic structure of a simulation experiment, define (minimally) required properties, suggest default values, or provide a list of values from which the user may select one. Experiment schemas are subdivided into modules that each concern a specific part of a simulation experiment specification. There are typical modules that are required for every simulation experiment, independently of the modeling and simulation domain. In particular, we identified as common constituents of simulation experiment specifications: *model* initialization, *simulator* initialization, and *observation*, which is in line with the studies of (Zeigler 1984; Waltemath et al. 2011; Ewald and Uhrmacher 2014). However, how these modules are filled with properties may look fundamentally different depending on the domain of modeling and simulation. For instance, in DES the model block typically refers to the parameters of agent-based models specified in a specialized modeling language (Macal and North 2005). On the other hand, FEM simulations typically require a geometric model, and a PDE model of the dynamics (Logg et al. 2012). Further, each modeling and simulation domain makes use of different simulation algorithms, and thus requires different inputs to parameterize the simulators. In addition, the observations differ between the various modeling and simulation domains, e.g., in DES we typically look at the dynamics of a system over time (Law and Kelton 2000), whereas in FEM simulations the focus lies due to decreased computational cost often on the frequency domain (Cangellaris 1996). The three modules *model*, *simulation*, and *observation* are mandatory for the specification of single simulations. However, if more complex experiments shall be specified that go beyond single simulation runs, further modules are required. In particular, the schema may be extended by a module for experiment design. It may be even desirable to divide this module further into separate submodules for different types of analyses such as sensitivity analysis, simulation-based optimization, or model checking, since experiment design and analysis method are closely related (Kleijnen 1998). During our case study in the Sections 3, 4, and 5, we will define basic schemas for two modeling and simulation domains, and extend them by a shared schema module for experiment design to illustrate the benefit of the modular organization. In particular, the modularization allows experiment schemas to be extended in a flexible way, e.g., by defining the basic modules for a new modeling and simulation domain, or by adding new choices of simulation algorithms, reporting formats, etc. to an existing schema, or by adding further modules for other simulation-based analysis methods and experiment designs that may be applicable across multiple domains of modeling and simulation.

A schema entry is a tuple (*Property, Description, Type, Choices, Defaults, Required*). *Property* is the unique identifier of the experiment property. *Description* is a verbal description of the property and its expected input, that for example may be displayed in the GUI. *Type* specifies which data type is expected, e.g., *String, Integer, or Real*. More specific types can be defined by adding Boolean expressions, e.g., the number of simulation replications should be an integer, greater than zero. *Choices* and *Defaults* have similar purposes as both provide a list of values from which a dropdown menu for the user can be built. However, *Defaults* is a list of (optional) default values, i.e., the user may select one but is not required to. Whereas *Choice* presents the user (or an automatic input extractor) with a fixed set of values, from which one has to be chosen. Finally, *Required* specifies whether for a certain property an input is *always* required (thereby making it a minimum requirement), *optional*, or *conditionally required*. Conditional requirements refer to the input values of other properties inside the same schema module, i.e., depending on a selection at an upper level, some subproperties will be required, yielding a hierarchical schema structure. For example, in the observation specification, the user may choose from different options of when to observe the simulation. Choices for this property are either a single point of time, or a time range with observation interval, or other conditions like steady state. If the user chooses the single observation point, another subproperty will be required, expecting a real value  $>0$ .

If any invalid inputs are found during the schema validation, an error feedback is sent to the user through the GUI, stating which inputs are either missing, or have an incorrect data type. If all type checks are passed, and no more required fields are empty an experiment object is created which is a name-value map that is valid according to the schema. It describes the experiment in terms of the used methods and algorithms, instead of concrete implementations. Thus, these first-level checks are backend-agnostic, and only provide a frame to capture common characteristics of certain classes of simulation experiments. As a result, the schema does not ensure that the experiment will be executable in a specific backend. Such backend-specific checks are forwarded to the second experiment generation level, i.e., the template rendering task.

### 2.3 Template Rendering

To transform the experiment object into an (executable) experiment specification in a target specification language of a backend we use a template-based approach. However, instead of using fixed templates that correspond to a particular experiment type - backend combination as in (Ruscheinski et al. 2018), we now have to account for the different input choices available to the user, since each choice potentially changes parts of the resulting experiment specification. Therefore, each schema property is mapped to a template fragment in the specification language of a selected modeling and simulation backend. Moreover, the abstract terms of the experiment schema (referring to methods or algorithms) have to be mapped to the concrete implementations of the selected backend. The template rendering process takes this backend mapping as input, as well as the experiment object. Using a template engine such as FreeMarker (FreeMarker 2019) it then joins together the necessary template fragments, and fills the template variables with data.

In general, a template fragment consists of a fixed text skeleton in the target specification language, and template variables  $\${property}$  that have to be filled with data from the experiment object, and are identified by the name of a schema property. For example, in the target language SESSL/ML-Rules the template fragment for observation at a specific point of time is `observeAt ( $\${atTime}$ )`, whereas the template fragment for observation of a time range is `observeAt (range ( $\${atRange[0]}$ ),  $\${atRange[1]}$ ,  $\${atRange[2]}$ )`. A schema entry may be mapped to template fragments of multiple target languages, but there may be at most one mapping per backend. Especially the common constituents of a simulation experiment, i.e., the minimally required information, have to be mapped to all the backends that are available for a modeling and simulation domain. In contrast, other more specific properties may only apply for one backend. A backend is defined as a combination of experiment language and input/modeling language, forming a modeling and simulation environment. Although some backends support similar types of simulations, and thus share a common schema, their implementations, used terminology, and available methods differ. For example, SESSL/ML-Rules and SESSL/ML3 (Warnke et al. 2015) both

support stochastic DES for cell-biological systems, and demographic systems respectively. The schema property *simulator* is translated to the template fragment `simulator = ${simulator}` for both backends. However, the selected simulation algorithm class *stochastic* is translated into the concrete implementations *StandardSimulator()* for SESSL/ML-Rules, and *NextReactionMethod()* for SESSL/ML3 to fill the template variable `${simulator}`. Apart from the information obtained directly from the user, some backends require additional syntax blocks, that do not belong to a specific schema module or property. Thus, for each backend a *master template* is provided that determines the order of the properties, and fills in the constant syntax elements. Further, some of the properties are associated with header information, e.g., if a verification method is requested, the template has to make sure, that all the necessary verification libraries are loaded, e.g., at the top of a SESSL/ML3 final experiment template the following line has to be added: `import sessl.verification._` (Reinhardt et al. 2018).

Finally, at the end of the template rendering task, the experiment specification may still contain syntactical or semantical errors. Therefore, backend-specific checks have to be performed using the backend's compiling unit. If all checks pass successfully, the final (executable) experiment specification is created, or otherwise the user is informed of the precise errors. The backend-specific checks may concern the concrete number format, or the syntax of an embedded language, i.e., one backend may expect temporal logic formulas to be expressed in metric interval temporal logic (MITL), while others expect linear temporal logic (LTL).

## 2.4 Experiment Execution

After input collection, validation, and schema mapping, the experiment specification is now complete and can be used in the selected backend. Firstly, it may be executed directly via a quickstart project, i.e., the experiment execution is started automatically as an additional step at the end of the generation process, and the simulation results are delivered to a location specified in the observation module of the experiment specification. A common interface of the backends allows for an automated retrieval of the simulation experiment results. When using this quickstart option, the generated experiment specification is kept internally, and is not presented to the user. Alternatively, the experiment specification may be presented to the user via the GUI, or exported as a file. From there it may be loaded into an external text editor, e.g., if the user wants to extend a generated NetLogo specification with custom R code (Thiele 2014). Or lastly, the experiment specification may be stored for the purpose of model documentation, for example, as part of an archive such as COMBINE (Bergmann et al. 2014), to enrich text-based documentations like ODD (Grimm et al. 2010), or as provenance in a scientific workflow (Oinn et al. 2004; Ludäscher et al. 2006).

Often, at the time of specification the user already has a backend in mind, and thus the current realization expects the user to specify a backend after the successful schema validation. However, sometimes the concrete backend may not be important, as long as the selected methods are supported, e.g., some statistical MITL model checker. Thus, future implementations may automatically choose a suitable backend based on the given inputs. This could be taken even one step further by not only selecting the backend automatically but also the experiment type, i.e., given some inputs from a model documentation, all possible analyses shall be performed such as some sensitivity analysis, model checking, etc., to get a quick overview of the model's behavior. This may also be combined with approaches for automatic method selection, e.g., the most efficient, or most accurate simulator (Leye et al. 2014).

## 3 A BASIC STOCHASTIC SIMULATION EXPERIMENT

To test our approach at work, we start with basic stochastic simulation experiments. Therefore, we first define a schema for stochastic DES by bringing together the required and optional information as discussed in the modeling and simulation literature (Law and Kelton 2000; Banks et al. 2010). As a result, the defined schema (Table 1) is applicable for many application domains of agent-based or individuals-based modeling and simulation, such as cell biology, demography, logistics, etc. In particular, the developed schema comprises the three essential parts of a simulation experiment specification, i.e., model initialization,

Table 1: Basic experiment schema for stochastic discrete-event simulation.

|             | Property            | Description                       | Type              | Choices         | Defaults   | Required                           |
|-------------|---------------------|-----------------------------------|-------------------|-----------------|------------|------------------------------------|
| Model       | modelPath           | Path to simulation model          | String            | -               | -          | always                             |
|             | configurationNames  | Parameter names                   | Array<String>     | -               | -          | optional                           |
|             | configurationValues | Parameter values                  | Array<Real>       | -               | -          | optional                           |
| Simulation  | simulator           | Choose simulation algorithm       | String            | stochastic, ... | stochastic | always                             |
|             | replications        | Simulation replications           | Integer, >0       | -               | 1          | required                           |
|             | stopCondition       | Types of stop conditions          | -                 | stopTime, ...   | -          | always                             |
|             | →stopTime           | Stop at specific point of time    | Real, >0          | -               | -          | if stopCondition=="stopTime"       |
|             | →stopExpression     | Stop based on simulation state    | BooleanExpression | -               | -          | if stopCondition=="stopExpression" |
| Observation | observation         | Observation information           | -                 | -               | -          | always                             |
|             | →obsExpressions     | Expression on model species       | Array<String>     | -               | -          | always                             |
|             | →obsAliases         | Alias for observation expression  | Array<String>     | -               | -          | optional                           |
|             | →obsTime            | Choose option for observation     | String            | atTime, ...     | -          | always                             |
|             | →atTime             | Observe at specific point of time | Real, >0          | -               | -          | if time=="atTime"                  |
|             | →range              | Observe time range and interval   | Array<Real>       | -               | -          | if time=="range"                   |
|             | outputFormat        | Choose reporting format           | String            | csv, ...        | -          | optional                           |

simulation initialization, and observation, which we established in our concept in Section 2. In the model initialization part, the schema asks for a path to a simulation model as well as a set of parameter configurations. In the simulation initialization part a type of simulator has to be chosen, e.g., a Gillespie style algorithm (Gillespie 1977), or a hybrid execution scheme (Haseltine and Rawlings 2002). Further, information about the stop condition, and the number of simulation replications is required. Since in DES we commonly observe the species of a system over time, in the observation part of the schema an observation time is required as well as an observation expression specifying which model species are of interest. Finally, the simulation output has to be recorded, and thus an output format may be chosen.

After we finished the schema definition for DES, we use it to collect and validate user input by iterating through the first two tasks of our experiment generation process. In our sample case study, the user aims to analyze a rule-based model of membrane binding processes specified in the file “TernaryComplexModel.mlrlj”. Thus, the user selects this file through the interactive GUI, and also specifies a parameter configuration. Further, the user specifies a stochastic simulation with 100 replications, evaluated until simulation time 30001. As observation they select the number of “C” species at stop time to be recorded under the variable “Complexes”, and stored in a CSV file. All the collected and validated input, i.e., the experiment object, are depicted in the upper part of Figure 2. From this experiment object, we then exemplarily generate an executable experiment specification for the backend SESSL/ML-Rules (Maus et al. 2011; Ewald and Uhrmacher 2014). The lower part of Figure 2 shows the completed experiment specification after all template fragments have been chained together, and the data from the experiment object has been inserted into the template variables by the template rendering step of our experiment generation. Although we demonstrated our experiment generation concepts using the backend SESSL/ML-Rules, many other tools from the area of DES may be exploited such as NetLogo/R (Thiele 2014), or SESSL/ML3 (Warnke et al. 2015).

#### 4 VERSATILY – A BASIC ELECTROMAGNETIC SIMULATION EXPERIMENT

Next, we evaluate how our concepts for experiment generation, and the experiences we gained for basic stochastic simulation experiments, can be transferred to the domain of FEM modeling and simulation. In general, the FEM allows us to solve partial differential equations numerically (Logg et al. 2012). To conduct FEM experiments, a geometric model as well as a physical model have to be provided. Further, for the simulation a solver type has to be chosen, e.g., the MUltifrontal Massively Parallel sparse direct Solver (mumps) for linear systems (Amestoy et al. 2000). The observation block usually specifies an output format, and some post-processing steps. In Table 2 we define a schema that reflects the general inputs for a FEM experiment in terms of model, simulation, and observation, analogously to the DES schema.


|  |   |   |
|--|---|---|
| Experiment Object                          | <pre> 1a modelPath="TernaryComplexModel.mlrlj" 2a configurationNames=["nR","nL","nX","nCells","kf", 3a "kr","ka","ku"] 4a configurationValues=[6e4, 1.05e9, 2.4e4, 1, 5.7e-13, 5a 1.7e-2, 5e-9, 8e-5] 6a simulator="stochastic" 7a replications=100 8a stopCondition="stopTime" 9a stopTime=30001                 </pre>  | <pre> 10a obsExpressions=["count("C)"] 11a obsAliases=["Complexes"] 12a obsTime="atTime" 13a atTime=30001 14a outputFormat="csv"                 </pre>   |
| Experiment Specification in SESSL/ML-Rules |    |   |
|  | <pre> 1b import sessl._ 2b import sessl.mlrules._ 3b execute( 4b   new Experiment with Observation with CSVOutput { 5b     model = "TernaryComplexModel.mlrlj" 6b     set("nR" &lt;- 6e4) 7b     set("nL" &lt;- 1.05e9) 8b     set("nX" &lt;- 2.4e4) 9b     set("nCells" &lt;- 1) 10b    set("kf" &lt;- 5.7e-13) 11b    set("kr" &lt;- 1.7e-2) 12b    set("ka" &lt;- 5e-9) 13b    set("ku" &lt;- 8e-5)                 </pre> | <pre> 14b   simulator = StandardSimulator() 15b   replications = 100 16b   stopTime = 30001 17b   observeAt(30001) 18b   observe("Complexes" ~ count("C")) 19b   withRunResult(writeCSV) 20b } 21b )                 </pre> |

Figure 2: Generating an experiment specification for a basic stochastic simulation experiment. The experiment object contains all collected user input encoded as name-value pairs. This information is transformed into an executable experiment specification in the language SESSL/ML-Rules. Blue - model initialization, orange - simulation initialization, green - observation, black - other backend-specific code.

However, in contrast to DES, where we saw a clear separation of concerns between model specification and experiment specification, for FEM simulations such a separation is not possible. In particular, the *model* block presents an amalgamation of model specification and experiment specification, and thus has been divided into two parts. Firstly, a geometric model has to be either selected or generated, and its parameter initializations have to be specified. Secondly, the physical model needs to be defined, i.e., the equations and the underlying laws of physics. For a strict separation of concerns this definition of the physical model would have to be removed from the experiment specification. However, due to its compact description using differential equations, a strict separation is not necessary. In contrast, models for DES are typically rather complex since their specification involves numerous degrees of freedom, e.g., in the definition of various agent types with attributes of arbitrary data types, behavioral rules, spatial properties of the modeled system, etc. As a consequence, one must use full-fledged modeling languages with formal semantics to obtain a concise description (Macal and North 2005). Further, a single schema for FEM simulations would not be of much use, since especially the physics part varies depending on the application domain. A common field of application for the FEM is electromagnetics (Meunier 2010). For these specific kinds of simulations the physical model block needs to contain the type of equations. Depending on the type of equations, material parameters such as conductivities and permittivities need to be provided. For a unique solution, proper boundary conditions need to be specified.

By defining the schemas for a) stochastic DES and b) FEM simulation applied to electromagnetics, we have seen that although they are two completely different domains, the general concepts about simulation experiments can be transferred. However, the overall versatility of the experiment schemas vary, i.e., while the DES schema covers a broader field of simulation, the FEM schema is already application-specific.

We apply the defined schema of Table 2 in a case study where the modeler wants to study the effect of electrical stimulation on cell cultures (Griffin et al. 2011; Budde et al. 2015). Again, the modeler first iterates through the input collection and validation phases of our experiment generation process. An excerpt of the resulting experiment object can be viewed in the upper part of Figure 3. From this experiment object we then exemplarily generate an executable experiment specification for the backend EMStimTools/YAML (Zimmermann 2019), which is a Python-based package aimed at linking free and

Table 2: Experiment schema for electromagnetic field simulations using the finite element method.

|                 | Property                       | Description  | Type                         | Choices                           | Required                          |
|-----------------|--------------------------------|--|------------------------------|-----------------------------------|-----------------------------------|
| Geom. Model     | studyName                      | Name of simulation study                             | String                       | -                                 | always                            |
|                 | dimensions                     | Number of dimensions                                 | Integer, >0, ≤ 3             | -                                 | always                            |
|                 | geometryType                   | Choose geometry type                                 | String                       | meshFile, geometryFile            | always                            |
|                 | →meshFile                      | Path to mesh file                                    | String                       | -                                 | if geometryType=="meshFile"       |
|                 | →geometryFile                  | Path to geometry file                                | String                       | -                                 | if geometryType=="geometryFile"   |
|                 | →subDomains<br>→geometryValues | List of geometry elements<br>List of geometry values | Array<String><br>Array<Real> | -<br>-                            | optional<br>optional              |
| Phys. Model     | physics                        | Choose a physical model                              | String                       | eqs, es, ...                      | always                            |
|                 | materials                      | Materials used in the model                          | Array<String>                | -                                 | always                            |
|                 | →conductivityValues            | Conductivities of the materials                      | Array<Real>                  | -                                 | if physics=="eqs" "es"            |
|                 | →permittivityValues            | Permittivities of the materials                      | Array<Real>                  | -                                 | if physics=="eqs"                 |
|                 | boundaryCondition              | Choose a boundary condition                          | String                       | dirichlet, ...                    | always                            |
|                 | →dirichlet                     | Dirichlet boundary condition                         | -                            | -                                 | if boundaryCondition=="dirichlet" |
| →boundaries     | List of boundaries             | Array<String>  | -                            | if boundaryCondition=="dirichlet" |                                   |
| →boundaryValues | List of boundary values        | Array<Real>  | -                            | if boundaryCondition=="dirichlet" |                                   |
| frequencies     | Frequency value                | Real   | -                            | if physics=="eqs"                 |                                   |
| Simulation      | solver                         | Choose a solver type                                 | String                       | mumps, ...                        | always                            |
|                 | element                        | Choose an element type                               | String                       | cg, ...                           | always                            |
|                 | degree                         | Degree of the polynomial                             | Integer                      | -                                 | always                            |
|                 | meshRefinement                 | Use iterative mesh refinement                        | -                            | -                                 | optional                          |
|                 | →selectedSubDomains            | Subdomains to be refined                             | Array<String>                | -                                 | optional                          |
|                 | →cycles                        | Number refinement steps                              | Array<Integer>               | -                                 | optional                          |
| Observation     | obsAlias                       | Alias for the observation                            | String                       | -                                 | optional                          |
|                 | obsPosition                    | Coordinates of the observed point                    | Array<Real>                  | -                                 | optional                          |
|                 | outputFormat                   | Choose an output format                              | String                       | xmdf, ...                         | optional                          |
|                 | postProcessing                 | Add options for postprocessing                       | -                            | -                                 | optional                          |
|                 | →properties                    | Calculation of derived properties                    | -                            | -                                 | optional                          |
|                 | →propertyNames                 | List of property names                               | Array<String>                | -                                 | optional                          |
|                 | →propertyValues                | List of expressions                                  | Array<Expression>            | -                                 | optional                          |
|                 | →mesh                          | Plot a mesh  | Boolean                      | -                                 | optional                          |
|                 | →submesh                       | Plot submeshes for given elements                    | Array<String>                | -                                 | optional                          |

open-source FEM tools like FEniCS (Logg et al. 2012) and SALOME (SALOME 2019), and show excerpts in the lower part of Figure 3. We would like to stress that by defining a common schema for all (electromagnetics) FEM simulations, we achieve a versatile experiment generation approach, in which the schemas may be reused to generate experiment specifications for other equivalent tools, e.g., the proprietary FEM platform COMSOL Multiphysics® (COMSOL 2019).

## 5 COMPOSABILITY - SENSITIVITY ANALYSIS

So far in our case studies we have looked at basic simulation experiments that involve single simulations. However, for a more comprehensive analysis of simulation models we are in the need of more complex experiment types that involve the execution of multiple simulations, and combine their results in an appropriate way. To conduct such complex experiments efficiently, using appropriate methods and experiment designs is crucial (Kleijnen 1998). Thus, in the following we will extend our previously defined schemas with a new schema module for experiment design. As an example for an experiment design we use a two-level full-factorial sensitivity analysis setup (Sanchez et al. 2018), that takes as inputs the factor names, as well as two values for each factor, i.e., the baseline case, and the pertubated case (see Table 3), and calculates the individual and interaction effects of the given factors (Saltelli et al. 2004). This rather simple schema may be easily extended with other experiment designs or methods such as metamodeling (Kleijnen 2009), or statistical model checking (Wald 1945) which is indicated in Table 3.



|  |  |   |
|--|--|---|
| Experiment Object                            | <pre> 1a study="study_griffin2011" 2a dimensions=3 3a geometryType="geometryFile" 4a geometryFile="griffin2011_salome.py" 5a subDomains=["w_air", "h_air", "r_el", "h_el", 6a   "gap", "r_dish", "h_dish", "t_dish"] 7a geometryValues=[1, .05, .04, 0.01, 0.002, 8a   0.0175, .01, 0.001] 9a physics="eqs" 10a materials=["dish", "medium", "air", "airgap"] 11a conductivityValues=[1e-14, 1.5, 1e-14, 1e-14] 12a permittivityValues=[2.5, 80., 1.0, 1.0] </pre> | <pre> 13a boundaryCondition="dirichlet" 14a boundaries=["UpperElectrode", 15a   "LowerElectrode"] 16a boundaryValues=[0.16, 0.0] 17a frequencies=15. 18a solver="mumps" 19a element="cg" 20a degree=2 21a obsAlias="solution" 22a obsPosition=[0.05, 0.017, 0.011] 23a outputFormat="xdmf" </pre> |
| Experiment Specification in EMStimTools/YAML | <pre> ... 3b SALOMEfile: 4b   griffin2011_salome.py ... 19b physics: 20b   EQS 21b materials: 22b   [dish, medium, air, airgap] 23b conductivity: 24b   dish : 1e-14 25b   medium : 1.5 ... 28b permittivity: 29b   dish : 2.5 30b   medium : 80. </pre>   | <pre> ... 33b boundaries: 34b   Dirichlet: 35b     UpperElectrode: 0.16 36b     LowerElectrode: 0.0 37b frequencies: 38b   15. 40b solver: 41b   linear_solver : mumps 42b element: 43b   CG ... 45b output: 46b   XDMF: yes ... </pre>   |

Figure 3: Generating an experiment specification for a finite element simulation of electromagnetic fields. Excerpts of the experiment object are shown in the upper part of the figure, excerpts of the generated executable experiment specification for the backend EMStimTools/YAML in the lower part respectively. Blue - geometric and physical model definition, orange - simulation initialization, green - observation.

Table 3: Experiment schema module defining structure and input for various experiment designs.

| Property                  | Description                     | ... | Choices  | Required  |
|---------------------------|---------------------------------|-----|--|---|
| experimentType            | Choose experiment type          | ... | statisticalModelChecking, sensitivityAnalysis, ... | optional  |
| →statisticalModelChecking | Choose stat. MC method          | ... | sequentialProbabilityRatioTest, ...                | if experimentType=="statisticalModelChecking"   |
| → ...                     | ...                             | ... | ...  | ...   |
| →sensitivityAnalysis      | Choose SA design                | ... | twoLevelFullFactorial, ...                         | if experimentType=="sensitivityAnalysis"        |
| →twoLevelFullFactorial    | Two-level full-factorial design | ... | -  | if sensitivityAnalysis=="twoLevelFullFactorial" |
| →factorNames              | Parameter names for level one   | ... | -  | if sensitivityAnalysis=="twoLevelFullFactorial" |
| →levelOneValues           | Parameter values for level one  | ... | -  | if sensitivityAnalysis=="twoLevelFullFactorial" |
| →levelTwoValues           | Parameter values for level two  | ... | -  | if sensitivityAnalysis=="twoLevelFullFactorial" |
| → ...                     | ...                             | ... | ...  | ...   |

Note, that the schema module for experiment design may be composed with both the DES schema and the FEM schema, and thus is reusable across the various modeling and simulation domains. That is, in both experiment objects the same schema properties are filled, but with different data. The experiment objects are then mapped to the backends SESSL/ML-Rules and EMStimTools/YAML by interweaving the basic experiment templates with templates for sensitivity analysis (for details see the accompanying material (Wilsdorf et al. 2019)).

## 6 RELATED WORK

Work on reporting standards is also aimed at representing simulation experiments in a tool-independent manner, e.g., the Minimum Information About a Simulation Experiment (MIASE) (Waltemath et al. 2011) in

the context of systems biology, or Strengthening the reporting of empirical simulation studies (STRESS) for the field of operational research and management sciences (Monks et al. 2019). However, those are neither formal nor executable. The Simulation Experiment Description Markup Language (SED-ML) (Waltemath et al. 2011) presents an implementation of the MIASE guidelines via an XML-based format. It is aimed at model documentation and at facilitating the reproduction of simulation results presented in systems biology publications. Consequently, diverse experiment types and their execution as required during conduction of a simulation study across different modeling and simulation domains are not in the focus.

In (Teran-Somohano et al. 2015) the specification of experiments is supported via a dynamic graphical user interface that is controlled based on an ontology (Lorscheid et al. 2012). Although the conceptual idea is similar as it involves the iterative experiment specification based on an ontology (or in our case experiment schema), they appear to currently only support the generation of XML files. Further, their ontology focuses on simple designs of experiments, and thus it becomes not clear what other domains or modeling and simulation approaches can be supported. In contrast, our approach supports widely different backends (as shown in Sections 3-5). In addition, the knowledge about simulation experiments is enhanced by default values, and requirements, which capture interdependencies in the specification of a simulation experiment. Further, the modular design of simulation experiments emphasizes the common characteristics of simulation experiments, as well as their differences, and facilitates the generation of sophisticated experiments. Both the enhancement and the modular design add to the growing body of knowledge about structure and ingredients of individual simulation experiments, e.g., (Ewald and Uhrmacher 2014; Yilmaz et al. 2016; Lorig et al. 2017; Sanchez et al. 2018; Ruschinski et al. 2018).

## 7 CONCLUSIONS

We introduced a two-level experiment generation process where at the first level an abstract experiment specification is generated that describes the simulation experiment in terms of the used methods and algorithms based on modularized experiment schemas. From this intermediate representation, at the second level we generate experiment specifications for concrete implementations by mapping schema properties to template fragments of a target backend. Our concepts for experiment generation were demonstrated using two different modeling and simulation domains as well as two corresponding backends. This application has shown that simulation experiment specifications go beyond specific tools since the required inputs are primarily determined by the overall modeling and simulation domain. Further, the versatility as well as composability of the approach could successfully be shown by moving from basic to more complex experiments. As a side effect, the experiment schemas provide guidelines for implementing new tools, and thus this paper also contributes to the creation of a body of knowledge for modeling and simulation. In future work, we will define mappings to additional backends. Also, major efforts will be directed to providing additional schemas to support a wide range of simulation experiments, e.g., for uncertainty quantification. Thereby, the structure and the (partly interdependent) ingredients of these simulation experiments will be made explicit, and thus we will contribute towards a better support for conducting simulation experiments and studies.

## ACKNOWLEDGMENTS

This work was funded by the research grants DFG UH 66/18 'GrEASE' and DFG CRC 1270 'Elaine'.

## REFERENCES

- Amestoy, P., I. Duff, and J.-Y. L'Excellent. 2000. "Multifrontal Parallel Distributed Symmetric and Unsymmetric Solvers". *Computer Methods in Applied Mechanics and Engineering* 184(2):501 – 520.
- Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2010. *Discrete-Event System Simulation*. 5th ed. Upper Saddle River, New Jersey: Prentice Hall.
- 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, M. Scharm, S. Soiland-Reyes, D. Waltemath, F. Yvon, and N. Le Novère. 2014.

- “COMBINE Archive and OMEX Format: one File to Share all Information to Reproduce a Modeling Project”. *BMC Bioinformatics* 15(1):369.
- Budde, K., J. Zimmermann, E. Neuhaus, M. Schröder, A. M. Uhrmacher, and U. van Rienen. 2015. “Requirements for Documenting Electrical Cell Stimulation Experiments for Replicability and Numerical Modeling”. In *41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. In Press.
- Cangellaris, A. C. 1996. “Frequency-Domain Finite Element Methods for Electromagnetic Field Simulation: Fundamentals, State of the Art, and Applications to EMI/EMC Analysis”. In *Proceedings of Symposium on Electromagnetic Compatibility*, 107–116. Piscataway, New Jersey: IEEE.
- COMSOL 2019. COMSOL Multiphysics® 5.4. <https://www.comsol.com>, accessed 5<sup>th</sup> April.
- Ewald, R., and A. M. Uhrmacher. 2014. “SESSL: A Domain-specific Language for Simulation Experiments”. *ACM Transactions on Modeling and Computer Simulation* 24(2):11:1–11:25.
- FreeMarker 2019. Apache FreeMarker Manual for FreeMarker 2.3.28. <https://freemarker.apache.org/docs/index.html>, accessed 5<sup>th</sup> April.
- Gillespie, D. T. 1977. “Exact Stochastic Simulation of Coupled Chemical Reactions”. *The Journal of Physical Chemistry* 81(25):2340–2361.
- Griffin, M., S. A. Iqbal, A. Sebastian, J. Colthurst, and A. Bayat. 2011. “Degenerate Wave and Capacitive Coupling Increase Human MSC Invasion and Proliferation While Reducing Cytotoxicity in an In Vitro Wound Healing Model”. *PLOS ONE* 6(8):1–14.
- 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.
- Haseltine, E. L., and J. B. Rawlings. 2002. “Approximate Simulation of Coupled Fast and Slow Reactions for Stochastic Chemical Kinetics”. *The Journal of Chemical Physics* 117(15):6959–6969.
- JSON 2019. ECMA-404 The JSON Data Interchange Standard. <http://www.json.org/>, accessed 5<sup>th</sup> April.
- JSON Schema 2019. JSON Schema Specification. <https://json-schema.org/specification.html>, accessed 5<sup>th</sup> April.
- Kleijnen, J. P. C. 1998. *Experimental Design for Sensitivity Analysis, Optimization, and Validation of Simulation Models*, Chapter 6, 173–223. New York: John Wiley & Sons, Inc.
- Kleijnen, J. P. C. 2009. “Kriging Metamodeling in Simulation: A Review”. *European Journal of Operational Research* 192(3):707–716.
- Law, A. M., and W. D. Kelton. 2000. *Simulation Modeling & Analysis*. 3rd ed. New York: McGraw-Hill, Inc.
- Leye, S., R. Ewald, and A. M. Uhrmacher. 2014. “Composing Problem Solvers for Simulation Experimentation: A Case Study on Steady State Estimation”. *PLOS ONE* 9(4):1–13.
- Logg, A., K.-A. Mardall, and G. N. Wells. 2012. *Automated Solution of Differential Equations by the Finite Element Method*. Berlin Heidelberg: Springer.
- Lorig, F., D. S. Leberherz, J. O. Berndt, and I. J. Timm. 2017. “Hypothesis-driven Experiment Design in Computer Simulation Studies”. In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D’Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 103:1–103:12. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- Lorscheid, I., B.-O. Heine, and M. Meyer. 2012. “Opening the ‘Black Box’ of Simulations: Increased Transparency and Effective Communication Through the Systematic Design of Experiments”. *Computational and Mathematical Organization Theory* 18(1):22–62.
- Ludäscher, B., I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. 2006. “Scientific workflow management and the Kepler system”. *Concurrency and Computation: Practice and Experience* 18(10):1039–1065.
- Macal, C. M., and M. J. North. 2005. “Tutorial on Agent-Based Modeling and Simulation”. In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 14–27. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- Maus, C., S. Rybacki, and A. M. Uhrmacher. 2011. “Rule-based Multi-level Modeling of Cell Biological Systems”. *BMC Systems Biology* 5(1):166.
- Meunier, G. 2010. *The Finite Element Method for Electromagnetic Modeling*, Volume 33. John Wiley & Sons.
- Monks, T., C. S. Currie, B. S. Onggo, S. Robinson, M. Kunc, and S. J. Taylor. 2019. “Strengthening the Reporting of Empirical Simulation Studies: Introducing the STRESS Guidelines”. *Journal of Simulation* 13(1):55–67.
- Oinn, T., M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. 2004. “Taverna: a tool for the composition and enactment of bioinformatics workflows”. *Bioinformatics* 20(17):3045–3054.
- Ören, T. I. 2005. “Toward the Body of Knowledge of Modeling and Simulation”. In *Proceedings of Interservice/Industry Training, Simulation, and Education Conference (IITSEC)*, 1–19. Arlington, VA: National Training and Simulation Association.
- Reinhardt, O., J. Hilton, T. Warnke, J. Bijak, and A. M. Uhrmacher. 2018. “Streamlining Simulation Experiments with Agent-Based Models in Demography”. *Journal of Artificial Societies and Social Simulation* 21(3):9.

- Ruscheinski, A., K. Budde, T. Warnke, P. Wilsdorf, B. C. Hiller, M. Dombrowsky, and A. M. Uhrmacher. 2018. "Generating Simulation Experiments based on Model Documentations and Templates". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 715–726. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- SALOME 2019. SALOME The Open Source Platform for Numerical Simulation. <https://www.salome-platform.org>, accessed 5<sup>th</sup> April.
- Saltelli, A., S. Tarantola, F. Campolongo, and M. Ratto. 2004. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. New York, NY: Halsted Press.
- Sanchez, S. M., P. J. Sánchez, and H. Wan. 2018. "Work Smarter, not Harder: a Tutorial on Designing and Conducting Simulation Experiments". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 237–251. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- Teran-Somohano, A., A. E. Smith, J. Ledet, L. Yilmaz, and H. Oğuztüzün. 2015. "A Model-driven Engineering Approach to Simulation Experiment Design and Execution". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 2632–2643. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- Thiele, J. C. 2014. "R marries NetLogo: Introduction to the RNetLogo Package". *Journal of Statistical Software* 58(2):1–41.
- Wald, A. 1945. "Sequential Tests of Statistical Hypotheses". *The Annals of Mathematical Statistics* 16(2):117–186.
- Waltemath, D., R. Adams, D. A. Beard, F. T. Bergmann, U. S. Bhalla, R. Britten, V. Chelliah, M. T. Cooling, J. Cooper, E. J. Crampin, A. Garny, S. Hoops, M. Hucka, P. Hunter, E. Klipp, C. Laibe, A. K. Miller, I. Moraru, D. Nickerson, P. Nielsen, M. Nikolski, S. Sahle, H. M. Sauro, H. Schmidt, J. L. Snoep, D. Tolle, O. Wolkenhauer, and N. Le Novère. 2011. "Minimum Information About a Simulation Experiment (MIASE)". *PLOS Computational Biology* 7(4):1–4.
- Waltemath, D., R. Adams, F. T. Bergmann, M. Hucka, F. Kolpakov, A. K. Miller, I. I. Moraru, D. Nickerson, S. Sahle, J. L. Snoep et al. 2011. "Reproducible Computational Biology Experiments with SED-ML-the Simulation Experiment Description Markup Language". *BMC systems biology* 5(1):198.
- Warnke, T., A. Steiniger, A. M. Uhrmacher, A. Klabunde, and F. Willekens. 2015. "ML3: a Language for Compact Modeling of Linked Lives in Computational Demography". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 2764–2775. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- Wilsdorf, P., J. Zimmermann, M. Dombrowsky, U. van Rienen, and A. M. Uhrmacher. 2019. Simulation Experiment Schemas - Beyond Tools and Simulation Approaches (Software Appendix). [https://doi.org/10.18453/rosdok\\_id00002465](https://doi.org/10.18453/rosdok_id00002465), accessed 1<sup>st</sup> December 2019.
- Yilmaz, L., S. Chakladar, and K. Doud. 2016. "The Goal-hypothesis-experiment Framework: A Generative Cognitive Domain Architecture for Simulation Experiment Management". In *Proceedings of the 2016 Winter Simulation Conference*, edited by T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, 1001–1012. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- Zeigler, B. P. 1984. *Multifaceted Modelling and Discrete Event Simulation*. San Diego, CA: Academic Press Professional, Inc.
- Zimmermann, J. 2019. EMStimTools. <https://github.com/j-zimmermann/EMStimTools/releases/tag/v0.1.2.dev0>, accessed 5<sup>th</sup> April.

## AUTHOR BIOGRAPHIES

**PIA WILSDORF** is a Ph.D. candidate in the Modeling and Simulation group at the University of Rostock. Her e-mail address is [pia.wilsdorf@uni-rostock.de](mailto:pia.wilsdorf@uni-rostock.de).

**JULIUS ZIMMERMANN** is a Ph.D. candidate in the Theoretical Electrical Engineering group at the University of Rostock. His e-mail address is [julius.zimmermann@uni-rostock.de](mailto:julius.zimmermann@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](mailto:marcus.dombrowsky@uni-rostock.de).

**URSULA VAN RIENEN** is professor at the Institute of General Electrical Engineering, University of Rostock, and chair of the Theoretical Electrical Engineering group. Her e-mail address is [ursula.van-rienen@uni-rostock.de](mailto:ursula.van-rienen@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](mailto:adelinde.uhrmacher@uni-rostock.de).